Importance sampling for weighted binary random matrices with specified margins

Matthew T. Harrison*
Jeffrey W. Miller*
Division of Applied Mathematics
Brown University
Providence, RI 02912

January 18, 2013

Abstract

A sequential importance sampling algorithm is developed for the distribution that results when a matrix of independent, but not identically distributed, Bernoulli random variables is conditioned on a given sequence of row and column sums. This conditional distribution arises in a variety of applications and includes as a special case the uniform distribution over zero-one tables with specified margins. The algorithm uses dynamic programming to combine hard margin constraints, combinatorial approximations, and additional non-uniform weighting in a principled way to give state-of-the-art results.

KEYWORDS: bipartite graph, conditional inference, permanent, Rasch model, uniform distribution

1 Introduction

Let Ω^* denote the set of $m \times n$ binary matrices with row sums $\boldsymbol{r} = (r_1, \ldots, r_m)$ and column sums $\boldsymbol{c} = (c_1, \ldots, c_n)$, and let $\boldsymbol{w} = (w_{ij}) \in [0, \infty)^{m \times n}$ be a given nonnegative matrix. Define the distribution P^* on $\{0,1\}^{m \times n}$ via

$$P^*(\boldsymbol{z}) = \frac{1}{\kappa} \prod_{ij} w_{ij}^{z_{ij}} \mathbb{1}\{\boldsymbol{z} \in \Omega^*\}, \qquad \kappa = \sum_{\boldsymbol{z} \in \Omega^*} \prod_{ij} w_{ij}^{z_{ij}}, \tag{1}$$

where $\mathbb{1}$ is the indicator function and where we assume $\kappa > 0$. P^* is the conditional distribution of an $m \times n$ array of independent Bernoulli random variables, say $\boldsymbol{B} = (B_{ij})$,

*Matthew T. Harrison is Assistant Professor of Applied Mathematics, Brown University, Providence, RI 02912 (email: Matthew_Harrison@Brown.edu) and Jeffrey W. Miller is a graduate student of Applied Mathematics, Brown University, Providence, RI 02912 (email: Jeffrey_Miller@Brown.edu). This work was supported in part by the National Science Foundation (NSF) grant DMS-1007593, the Defense Advanced Research Projects Agency (DARPA) contract FA8650-11-1-7151, and, while MTH was in the Department of Statistics at Carnegie Mellon University, by the NSF grant DMS-0240019 and the National Institutes of Health (NIH) grant NIMH-2RO1MH064537. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF, the NIH, or DARPA. The authors thank Sam Kou for sharing his code for approximating α-permanents.

with $\mathbb{P}(B_{ij} = 1) = w_{ij}/(1 + w_{ij})$ given the margins $\boldsymbol{r}$ and $\boldsymbol{c}$, where $\mathbb{P}$ denotes probability. This paper describes an importance sampling algorithm that can be used for Monte Carlo approximation of probabilities and expectations under P^* and also for Monte Carlo approximation of κ. After a preprocessing step, sampling from our proposal distribution requires $O(md)$ operations per matrix, where $d = \sum_j c_j = \sum_i r_i$ is the total number of ones in the matrix.

We are not aware of any existing importance sampling algorithms that permit practical inference under P^*, although many special cases have been studied in the literature. For example, if $\boldsymbol{w} \equiv 1$ then P^* is the uniform distribution over zero-one tables with specified margins, or equivalently, the uniform distribution over bipartite graphs with specified degree sequence. For square matrices, if $\boldsymbol{w}$ is identically one except with a zero diagonal, then P^* corresponds to the uniform distribution over directed graphs with specified degrees. And if $\boldsymbol{r} = \boldsymbol{c} \equiv 1$, then P^* is a distribution over weighted permutation matrices and κ is the permanent of $\boldsymbol{w}$. Empirically, our algorithm outperforms all existing importance sampling algorithms in these special cases. Although our algorithm works well for most examples arising in practice, performance depends on $\boldsymbol{r}$, $\boldsymbol{c}$, and $\boldsymbol{w}$. Highly irregular margins or highly variable $\boldsymbol{w}$, particularly many zero entries in $\boldsymbol{w}$, tend to cause poor performance.

P^* factors in such a way that we need only focus on the distribution, say P, of the first column. The columns are sampled sequentially, with each successive column viewed as the first column of a smaller matrix with updated margins based on the previously sampled columns. We decompose the structure of P into margin constraints, combinatorial factors, and non-uniform weighting terms, combine approximations of these terms in a principled way, and then use a dynamic programming algorithm to exactly and efficiently sample from the resulting proposal distribution Q for the first column. Sequentially sampling columns in this way defines a proposal distribution Q^* for the whole matrix. This strategy for algorithm design works well for many similar problems, including symmetric matrices and nonnegative integer-valued matrices, each of which will be described elsewhere owing to space constraints. It seems likely that the design principles used for our approach are applicable much more broadly.

2 Motivating applications

2.1 Conditional inference for graphs and tables

Let $\boldsymbol{B} \in \{0, 1\}^{m \times n}$ be a matrix of independent Bernoulli random variables with

$$\operatorname{logit} \mathbb{P}(B_{ij} = 1) = \alpha_i + \beta_j + \textstyle\sum_k \theta_k \xi_{kij}, \tag{2}$$

where $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, and $\boldsymbol{\theta}$ are parameters, perhaps with constraints to ensure identifiability, and $\boldsymbol{\xi}$ is a collection of observed covariates. Models of this form arise, for example, in educational testing, where B_{ij} indicates whether or not subject i responded correctly to question j. If $\boldsymbol{\theta} \equiv 0$, then the model reduces to the classical Rasch model (Rasch, 1960, 1961). Otherwise, it is an extension of the Rasch model to include item-specific covariate effects, such as each subject's prior exposure to the content being tested in each question. This model is also a simple version of models used for the analysis of network data (c.f., Holland & Leinhardt, 1981; Fienberg et al., 1985; Goldenberg et al., 2010) where $\boldsymbol{B}$ is the adjacency matrix of a directed graph, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ allow for degree heterogeneity, and $\boldsymbol{\xi}$ is a collection of edge-specific

covariates. For example, for social network data we might have that B_{ij} indicates whether subject i reported subject j as a friend, α_i controls the relative propensity for subject i to report friends, β_j controls the relative propensity for subject j to be reported as a friend, and ξ_{kij} indicates whether the relationship between subject i and j is of type k.

In both of these examples, if $\boldsymbol{\theta}$ is the only parameter of interest, then the nuisance parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ complicate inference and can be removed by conditioning on the row and column sums of $\boldsymbol{B}$ (e.g., Cox, 1958; Holland & Leinhardt, 1981; Mehta & Patel, 1995; Harrison, 2012). Conditioning also results in inferential procedures that are robust to modeling assumptions, implicit in (2), about the distribution of the margins. The resulting conditional model that drives inference is exactly $P^* = P^*_\theta$ with $w_{ij} = \exp(\sum_k \theta_k \xi_{kij})$, perhaps with the additional constraint that $w_{ii} = 0$ in the case of network data. The conditional model is a natural exponential family in $\boldsymbol{\theta}$ with no nuisance parameters, but with an intractable normalization constant $\kappa = \kappa_\theta$. Harrison (2012, Example 4.2) provides details about an importance sampling approach to exact conditional inference for this model. The example there is based on a preliminary version of the algorithm presented here.

Other approaches to conditional inference in this setting include exhaustive enumeration, such as the algorithms for conditional logistic regression in Stata (StataCorp, 2009) and LogXact (Cytel, 2010), Markov chain Monte Carlo approaches, such as the `elrm` R package (Zamar et al., 2007), and analytic approximations, such as the `cond` R package (Brazzale, 2005; Brazzale & Davison, 2008), none of which are practical for larger matrices and/or multivariate $\boldsymbol{\theta}$. Approximation of κ was considered in Barvinok (2010b).

2.2 The uniform distribution and model validation

If $\boldsymbol{w} \equiv 1$, or more generally, if $w_{ij} = \exp(\alpha_i + \beta_j)$ for real-valued $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, then P^* is the uniform distribution on Ω^*. The uniform distribution can be used for testing if $\boldsymbol{\theta} \equiv 0$ in model (2), or equivalently, for model validation of (2) specialized to the case of $\boldsymbol{\theta} \equiv 0$. Prominent examples include goodness-of-fit tests for the Rasch model (e.g., Rasch, 1960, 1961; Ponocny, 2001; Chen & Small, 2005; Chen et al., 2005) and for random bipartite graphs and directed graphs without reciprocity (e.g., Wasserman, 1977; Holland & Leinhardt, 1981; Snijders, 1991).

The uniform distribution over Ω^* also plays a central role in testing for the presence of interactions in co-occurrence tables, particularly co-occurrence tables arising in ecology, where B_{ij} indicates the existence of species i in location j (e.g., Connor & Simberloff, 1979; Snijders, 1991; Gotelli, 2000; Chen et al., 2005). In these contexts, the uniform distribution subject to the margin totals is taken as a null hypothesis of no interaction among species (without necessarily assuming model (2)). Our original motivation for developing these algorithms came from a similar problem in neuroscience where B_{ij} indicated whether neuron i produced an action potential in time bin j, and the uniform distribution was taken as a null hypothesis of a lack of interaction among the neurons. This can be viewed as an example of conditional testing for multivariate binary time series. In that example, $mn \approx 10^8$ and existing algorithms in the literature were not practical.

Besides the many statistical applications, when $\boldsymbol{w} \equiv 1$, the normalization constant κ is the number of binary matrices with specified margins, a topic of enduring interest in theoretical computer science (e.g., Kannan et al., 1999; Jerrum et al., 2004; Bezáková et al., 2007) and combinatorial approximation (e.g., Békéssy et al., 1972; McKay, 1984; Greenhill et al., 2006; Canfield et al., 2008; Barvinok, 2010a). Importance sampling algorithms for P^* can

be used to provide efficient approximations of κ (Blanchet, 2009).

Monte Carlo sampling algorithms for the uniform distribution have been developed by many authors (e.g., Besag & Clifford, 1989; McKay & Wormald, 1990; Snijders, 1991; Rao et al., 1996; Chen et al., 2005; Blanchet, 2009; Bezáková et al., 2007; Chen, 2007; Verhelst, 2008; Bayati et al., 2010). The approach here was inspired by the importance sampling algorithm in Chen et al. (2005), but provides a more principled method for algorithm design that leads to substantial improvements in the uniform case and that also extends to the non-uniform case.

2.3 Permanents and permanental processes

If $\boldsymbol{w}$ is square and $\boldsymbol{r} = \boldsymbol{c} \equiv 1$, that is, if Ω^* is the set of permutation matrices, then κ is the permanent of $\boldsymbol{w}$, also of enduring interest in theoretical computer science (e.g., Valiant, 1979; Jerrum et al., 2004). A variety of generalizations of permanents and determinants can be expressed as $\kappa\mu$, where $\mu = \mathbb{E}(h(\boldsymbol{Z}))$ for some function h, where $\mathbb{E}$ denotes expected value, and where $\boldsymbol{Z}$ has distribution P^* (e.g., Littlewood, 1950; Vere-Jones, 1988, 1997; Diaconis & Evans, 2000). In principle, the algorithms here could be used to approximate the value of any of these objects, but the practicality of this approach depends heavily on h. For example, the α-permanent (Vere-Jones, 1988, 1997) is

$$\mathrm{per}_\alpha(\boldsymbol{w}) = \kappa\, \mathbb{E}(\alpha^{\mathrm{cyc}(\boldsymbol{Z})}), \tag{3}$$

where $\alpha \in \mathbb{R}$ and $\mathrm{cyc}(\boldsymbol{z})$ is the number of disjoint cycles in the permutation corresponding to $\boldsymbol{z}$. The case $\alpha = 1$ corresponds to the permanent of $\boldsymbol{w}$, and the case $\alpha = -1$ corresponds to $(-1)^n$ times the determinant of $\boldsymbol{w}$. Permanents and α-permanents arise in probability, statistics, and statistical physics in connection to permanental processes and random fields (e.g., Macchi, 1975; Diaconis & Evans, 2000; Shirai & Takahashi, 2003; McCullagh & Møller, 2006; Kou & McCullagh, 2009) and the distribution of order statistics (Vaughan & Venables, 1972; Bapat & Beg, 1989). Our approach is often effective for approximating (3) when $\alpha > 0$ and $|\log \alpha|$ is small.

3 Algorithm design

3.1 The target distribution for the first column

For a matrix $\boldsymbol{z} = (z_{ij})$ we use $\boldsymbol{z}^j$ to denote the jth column of $\boldsymbol{z}$, we use $\boldsymbol{z}^{j:k}$ to denote the submatrix formed from columns $j, \ldots, k$, we use $\boldsymbol{R}(\boldsymbol{z}) = (R_i(\boldsymbol{z}))$ to denote the column vector of row sums defined by $R_i = \sum_j z_{ij}$, and we use $\boldsymbol{C}(\boldsymbol{z}) = (C_j(\boldsymbol{z}))$ to denote the row vector of column sums defined by $C_j = \sum_i z_{ij}$. Fix the size of the matrix, $m \times n$, the weights $\boldsymbol{w}$, and the margins, $\boldsymbol{r}$ and $\boldsymbol{c}$, and let $\boldsymbol{Z}$ have distribution P^* defined in (1) with $\Omega^* = \{\boldsymbol{z} \in \{0,1\}^{m\times n} : \boldsymbol{R}(\boldsymbol{z}) = \boldsymbol{r},\ \boldsymbol{C}(\boldsymbol{z}) = \boldsymbol{c}\}$.

To sample from P^* we need only design a generic algorithm (generic in $m, n, \boldsymbol{r}, \boldsymbol{c}, \boldsymbol{w}$) for sampling from the distribution P of the first column, namely,

$$P(\boldsymbol{x}) = \mathbb{P}(\boldsymbol{Z}^1 = \boldsymbol{x}).$$

The reason is that the conditional distribution of $\boldsymbol{Z}^{2:n}$ given $\boldsymbol{Z}^1$ has the same form as P^* in (1), but with different parameters. The size of the matrix is now $m \times (n-1)$, the row sums

are updated to $\boldsymbol{r} - \boldsymbol{R}(\boldsymbol{Z}^1)$, the column sums are updated to $\boldsymbol{c}^{2:n}$, and the weight matrix is updated to $\boldsymbol{w}^{2:n}$. Once we have sampled $\boldsymbol{Z}^1$, then we can update these parameters and effectively start over, treating the second column of $\boldsymbol{Z}$ like it was the first column of the new, updated problem, and then continuing sequentially until we have sampled the entire matrix. This is the same sequential strategy suggested by Chen et al. (2005). The supplementary material (located at the end of this document) contains a more detailed description of this column-wise factorization.

Henceforth, our target distribution is P, the distribution of $\boldsymbol{Z}^1$. Let $\boldsymbol{Y}$ be a random matrix chosen uniformly over Ω^*, let Ω denote the support of $\boldsymbol{Y}^1$, namely,

$$\Omega = \{\boldsymbol{x} \in \{0,1\}^{m\times 1} : \boldsymbol{x} = \boldsymbol{z}^1, \boldsymbol{z} \in \Omega^*\},$$

and for $\boldsymbol{x} \in \Omega$ define

$$U(\boldsymbol{x}) = \mathbb{P}(\boldsymbol{Y}^1 = \boldsymbol{x}), \qquad V(\boldsymbol{x}) = \mathbb{E}\Big(\prod_{ij} w_{ij}^{Y_{ij}} \Big| \boldsymbol{Y}^1 = \boldsymbol{x}\Big).$$

It is straightforward to verify that

$$P(\boldsymbol{x}) \propto U(\boldsymbol{x})V(\boldsymbol{x})\mathbb{1}\{\boldsymbol{x} \in \Omega\} \tag{4}$$

for $\boldsymbol{x} \in \{0,1\}^{m\times 1}$. This factorization conceptually isolates the hard margin constraints, Ω, the combinatorics, U, and the non-uniform weighting, V. Although the separation is clearly artificial, it is useful to treat each of these factors separately when developing a proposal distribution.

3.2 The proposal distribution for the first column

Motivated by the factorization in (4), we consider proposal distributions for the first column of the form

$$Q(\boldsymbol{x}) \propto \tilde{U}(\boldsymbol{x})\tilde{V}(\boldsymbol{x})\mathbb{1}\{\boldsymbol{x} \in \tilde{\Omega}\},$$

where $\tilde{U}$ and $\tilde{V}$ are approximations of U and V, respectively, that factor according to

$$\tilde{U}(\boldsymbol{x}) \propto \prod_{i=1}^m u_i^{x_i}, \qquad \tilde{V}(\boldsymbol{x}) \propto \prod_{i=1}^m v_i^{x_i}$$

for some $\boldsymbol{u}, \boldsymbol{v} \in [0,\infty)^{m\times 1}$, and where $\tilde{\Omega}$ is of the form

$$\tilde{\Omega} = \big\{\boldsymbol{x} \in \{0,1\}^{m\times 1} : x_{\pi_i} \in \mathcal{A}_i,\ \textstyle\sum_{\ell=1}^i x_{\pi_\ell} \in \mathcal{B}_i,\ i = 1,\dots,m\big\} \tag{5}$$

for some permutation $\boldsymbol{\pi} = (\pi_1,\dots,\pi_m)$ of $(1,\dots,m)$ and some subsets $\boldsymbol{\mathcal{A}} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_m \subseteq \{0,1\}^m$ and $\boldsymbol{\mathcal{B}} = \mathcal{B}_1 \times \cdots \times \mathcal{B}_m \subseteq \{0,1,\dots,c_1\}^m$. Combining these approximations creates a proposal distribution of the form

$$Q(\boldsymbol{x}) \propto \prod_{i=1}^m u_i^{x_i} v_i^{x_i} \mathbb{1}\big\{x_{\pi_i} \in \mathcal{A}_i,\ \textstyle\sum_{\ell=1}^i x_{\pi_\ell} \in \mathcal{B}_i\big\} \qquad (\boldsymbol{x} \in \{0,1\}^{m\times 1}). \tag{6}$$

Any proposal distribution of this form permits fast, exact sampling and evaluation using $O(mc_1)$ operations; see Section 3.3. The challenge is to find easily computable choices of

$\boldsymbol{u}$, $\boldsymbol{v}$, $\boldsymbol{\pi}$, $\boldsymbol{\mathcal{A}}$, and $\boldsymbol{\mathcal{B}}$ such that Q is a good approximation to the target P. Fortunately, this seems to be possible in many cases; see Section 4.

For importance sampling to work, the support of Q, which is a subset of $\tilde{\Omega}$, must contain the support of P, which is a subset of Ω. When $\boldsymbol{w}$ has no zero entries, we require $\boldsymbol{u}$ and $\boldsymbol{v}$ to be positive and we engineer $\tilde{\Omega}$ to exactly coincide with Ω so that P and Q both have support Ω; see Section 4.1. When $\boldsymbol{w}$ does have zero entries, we modify $\boldsymbol{v}$ and $\tilde{\Omega}$ to exclude certain elements in Ω, but only elements that are not in the support of P. This ensures that the support of Q contains the support of P, but the supports may no longer be identical. In this case, if the importance sampling algorithm generates a column that is not in the support of P, then as it sequentially generates additional columns it will eventually try to create a Q that is identically zero, indicating that no assignment of the current column simultaneously satisfies the margin constraints and has positive weight. At this point the algorithm can assign an importance weight of zero and terminate. Certain patterns of zero weights make the algorithm highly inefficient because the algorithm rarely terminates with a nonzero importance weight. In the supplementary material we discuss alternative choices of $\boldsymbol{v}$ and $\tilde{\Omega}$ that are more efficient for certain patterns of particular interest, including the special case of zeros only on the diagonal.

3.3 Efficient sampling and evaluation of the proposal

Let $\boldsymbol{X} \in \{0,1\}^{m\times 1}$ have a distribution Q that factors according to (6) above for some $\boldsymbol{u}$, $\boldsymbol{v}$, $\boldsymbol{\pi}$, $\mathcal{A}$, and $\mathcal{B}$. Define the permuted partial sums $\boldsymbol{S} \in \{0,\dots,c_1\}^{m\times 1}$ according to $S_i = \sum_{\ell=1}^{i} X_{\pi_\ell}$ for each i, and note that $\boldsymbol{X}$ and $\boldsymbol{S}$ are in bijective correspondence. The distribution of $\boldsymbol{S}$ factors according to

$$\mathbb{P}(\boldsymbol{S}=\boldsymbol{s}) \propto \prod_{i=1}^{m} h_i(s_{i-1}, s_i) \tag{7}$$

for $h_i(s_{i-1}, s_i) = u_{\pi_i}^{s_i - s_{i-1}} v_{\pi_i}^{s_i - s_{i-1}} \mathbb{1}\{s_i - s_{i-1} \in \mathcal{A}_i,\ s_i \in \mathcal{B}_i\}$, where here and below we define $S_0 = s_0 = 0$ for notational convenience.

The factorization in (7) implies that $\boldsymbol{S}$ is a Markov chain. If we were given the standard Markov chain representation

$$\mathbb{P}(\boldsymbol{S}=\boldsymbol{s}) = \prod_{i=1}^{m} \mathbb{P}(S_i = s_i \big| S_{i-1} = s_{i-1}), \tag{8}$$

then generating a random observation of $\boldsymbol{S}$ would be trivial. It is known that dynamic programming can be used to convert from Gibbs random field representations like (7) into Bayesian network representations like (8); see, e.g., Frey (1998). The next theorem, which is straightforward to verify (cf. Harrison & Geman, 2009), summarizes dynamic programming in this context.

Theorem 1. *Let $(S_0, S_1, \dots, S_m)$ be a sequence of random variables where each S_i takes values in the finite set D_i and where $D_0 = \{0\}$. Suppose there exists a sequence of functions $h_i : D_{i-1} \times D_i \mapsto [0,\infty)$ for $i = 1,\dots,m$ such that the distribution of $(S_1,\dots,S_m)$ can be expressed as*

$$\mathbb{P}\big(S_1 = s_1, \dots, S_m = s_m\big) \propto \prod_{i=1}^{m} h_i(s_{i-1}, s_i).$$

Recursively define $g_m(s_{m-1}, s_m) = h_m(s_{m-1}, s_m)$ *and*

$$g_i(s_{i-1}, s_i) = h_i(s_{i-1}, s_i) \sum_{s_{i+1} \in D_{i+1}} g_{i+1}(s_i, s_{i+1}) \quad (i = 1, \ldots, m-1),$$

where each g_i *is defined over* $D_{i-1} \times D_i$*. Then* $S_0, \ldots, S_m$ *is a Markov chain and*

$$\mathbb{P}\big(S_i = s_i \big| S_{i-1} = s_{i-1}\big) = \frac{g_i(s_{i-1}, s_i)}{\sum_{t \in D_i} g_i(s_{i-1}, t)} \quad (i = 1, \ldots, m).$$

In the present context, $S_i \in \mathcal{B}_i \subseteq \{0, \ldots, c_1\}$ for each i, so the algorithm described in Theorem 1 for converting from (7) to (8) requires at most $O(mc_1^2)$ operations. In fact, since in the present situation we have $h_i(s_{i-1}, s_i) = 0$ for $s_i - s_{i-1} \notin \{0, 1\}$, implying the same for g_i, this yields an algorithm that requires $O(mc_1)$ operations. Instead of representing all $(c_1 + 1)^2$ combinations of (s_{i-1}, s_i), we represent only the $2c_1 + 1$ feasible combinations. Once the representation in (8) is computed, generating a random observation $\boldsymbol{X}$ from Q or evaluating $Q(\boldsymbol{x})$ at any $\boldsymbol{x}$ takes $O(m)$ operations.

4 Specification of components

4.1 Margin constraints

Here we discuss the construction of $\tilde{\Omega}$. In particular, the next theorem shows how to ensure that $\tilde{\Omega} = \Omega$ for easily computable choices of $\boldsymbol{\pi}$, $\boldsymbol{\mathcal{A}}$, and $\boldsymbol{\mathcal{B}}$.

Theorem 2. *(Chen et al., 2005) Assume* $\Omega^* \neq \emptyset$*. Choose* $\boldsymbol{\pi}$ *so that* $r_{\pi_1} \geq \cdots \geq r_{\pi_m}$*. For each* $i = 1, \ldots, m$*, define*

$$\mathcal{A}_i = \begin{cases} \{0\} & (r_{\pi_i} = 0); \\ \{0, 1\} & (0 < r_{\pi_i} < n); \\ \{1\} & (r_{\pi_i} = n), \end{cases} \qquad \mathcal{B}_i = \begin{cases} \{\max\{0, b_i\}, \ldots, c_1\} & (i < m); \\ \{c_1\} & (i = m), \end{cases}$$

for $b_i = \sum_{\ell=1}^{i} (r_{\pi_\ell} - \sum_{j=2}^{n} \mathbb{1}\{c_j \geq \ell\})$*. Define* $\tilde{\Omega}$ *according to* (5)*. Then* $\tilde{\Omega} = \Omega$*.*

It is instructive to see how these choices of $\boldsymbol{\pi}$, $\boldsymbol{\mathcal{A}}$, and $\boldsymbol{\mathcal{B}}$ ensure that $\tilde{\Omega} \supseteq \Omega$, which is the primary requirement for importance sampling. The Gale–Ryser conditions (Gale, 1957; Ryser, 1957) state that there is a binary matrix with margins $\boldsymbol{r} \in \{0, \ldots, n\}^{m \times 1}$ and $\boldsymbol{c} \in \{0, \ldots, m\}^{1 \times n}$ if and only if $\sum_i r_i = \sum_j c_j$ and

$$\sum_{\ell=1}^{i} r_{\pi_\ell} \leq \sum_{\ell=1}^{i} \sum_{j} \mathbb{1}\{c_j \geq \ell\} \quad \text{for all } i = 1, \ldots, m-1,$$

where the permutation $\boldsymbol{\pi}$ is chosen so that $r_{\pi_1} \geq \cdots \geq r_{\pi_m}$. It is straightforward to see that $\boldsymbol{x} \in \{0, 1\}^{m \times 1}$ will be in Ω exactly when there is a way to fill out the remaining $n - 1$ columns of the binary matrix that obey the updated margins after accounting for $\boldsymbol{x}$. In other words, $\boldsymbol{x} \in \Omega$ exactly when $\boldsymbol{r} - \boldsymbol{x}$ and $\boldsymbol{c}^{2:n}$ satisfy the Gale–Ryser conditions for the margins of an $m \times (n-1)$ binary matrix.

The set $\boldsymbol{\mathcal{A}}$ is chosen so that $\boldsymbol{x} \in \boldsymbol{\mathcal{A}}$ if and only if $\boldsymbol{r} - \boldsymbol{x} \in \{0, \ldots, n-1\}^{m \times 1}$. The set $\mathcal{B}_m$ is chosen so that $\sum_{i=1}^{m} x_i \in \mathcal{B}_m$ if and only if $\sum_{i=1}^{m} r_i - x_i = \sum_{j=2}^{n} c_j$. This is

equivalent to enforcing the column sum $\sum_{i=1}^{m} x_i = c_1$. Choosing the permutation $\boldsymbol{\phi}$ so that $r_{\phi_1} - x_{\phi_1} \geq \cdots \geq r_{\phi_m} - x_{\phi_m}$, the remaining Gale–Ryser conditions are that

$$\sum_{\ell=1}^{i}(r_{\phi_\ell} - x_{\phi_\ell}) \leq \sum_{\ell=1}^{i}\sum_{j\geq 2} \mathbb{1}\{c_j \geq \ell\} \quad \text{for all } i = 1, \ldots, m-1, \tag{9}$$

which implies that

$$\sum_{\ell=1}^{i}(r_{\pi_\ell} - x_{\pi_\ell}) \leq \sum_{\ell=1}^{i}\sum_{j\geq 2} \mathbb{1}\{c_j \geq \ell\} \quad \text{for all } i = 1, \ldots, m-1, \tag{10}$$

since the permutation $\boldsymbol{\phi}$ makes the left side as large as possible. Solving (10) for $\sum_{\ell=1}^{i} x_{\pi_\ell}$ gives the bounds encoded in the $\mathcal{B}_i$ and shows that $\tilde{\Omega} \supseteq \Omega$.

We cannot use the permutation $\boldsymbol{\phi}$ in the construction of Q because $\boldsymbol{\phi}$ depends on $\boldsymbol{x}$, however, Chen et al. (2005) further prove that (9) and (10) are in fact equivalent, which means we are in the ideal situation where $\Omega = \tilde{\Omega}$. (Although they made use of the factorization in Theorem 2, their proposal distributions were not of the form in (6), except in the special case where $\Omega = \{\boldsymbol{x} \in \{0,1\}^{m\times 1} : \sum_i x_i = c_1\}$.) Furthermore, Chen (2007) provides an extension of Theorem 2 for the case where, in addition to the margin constraints, Ω^* also enforces a fixed pattern of structural zeros for which there is at most one structural zero in each row and column. This includes the important special case of adjacency matrices of directed graphs; see supplementary material.

4.2 Combinatorial approximations

Here we discuss approximation of U by $\tilde{U}$. Define

$$N_{m,n}(\boldsymbol{r}, \boldsymbol{c}) = \left|\{\boldsymbol{z} \in \{0,1\}^{m\times n} : \boldsymbol{R}(\boldsymbol{z}) = \boldsymbol{r},\ \boldsymbol{C}(\boldsymbol{z}) = \boldsymbol{c}\}\right|$$

to be the number of $m \times n$ binary matrices with row sums $\boldsymbol{r}$ and column sums $\boldsymbol{c}$. We have

$$U(\boldsymbol{x}) = \mathbb{P}(\boldsymbol{Y}^1 = \boldsymbol{x}) = \frac{N_{m,n-1}(\boldsymbol{r} - \boldsymbol{x}, \boldsymbol{c}^{2:n})}{N_{m,n}(\boldsymbol{r}, \boldsymbol{c})},$$

(where as before, $\boldsymbol{Y}$ is uniform over Ω^*) and we desire an approximation of the form

$$U(\boldsymbol{x}) = \frac{N_{m,n-1}(\boldsymbol{r} - \boldsymbol{x}, \boldsymbol{c}^{2:n})}{N_{m,n}(\boldsymbol{r}, \boldsymbol{c})} \approx \gamma \prod_{i=1}^{m} u_i^{x_i}, \tag{11}$$

where γ is an irrelevant positive constant.

Temporarily pretending that (11) is accurate for any $\boldsymbol{x} \in \{0,1\}^{m\times 1}$, we have

$$u_i \approx \frac{U(\boldsymbol{I}^i)}{U(\boldsymbol{0})} = \frac{N_{m,n-1}(\boldsymbol{r} - \boldsymbol{I}^i, \boldsymbol{c}^{2:n})}{N_{m,n-1}(\boldsymbol{r}, \boldsymbol{c}^{2:n})},$$

where $\boldsymbol{I}^i$ is the ith column of the $m \times m$ identity matrix $\boldsymbol{I}$. We cannot use this directly, since it is trying to evaluate U outside of Ω, and, furthermore, computationally efficient procedures for evaluating $N_{m,n-1}$ are not available. Nevertheless, it suggests using

$$u_i = \tilde{N}_{m,n-1}(\boldsymbol{r} - \boldsymbol{I}^i, \boldsymbol{c}^{2:n}) / \tilde{N}_{m,n-1}(\boldsymbol{r}, \boldsymbol{c}^{2:n}) \qquad (i = 1, \ldots, m) \tag{12}$$

for an approximation $\tilde{N}$ of N that extends smoothly to invalid margins.

Several asymptotic approximations for N have appeared in the literature and could be used for $\tilde{N}$. For example, Canfield et al. (2008, Theorem 1) suggest the following, which we write asymmetrically with respect to $\boldsymbol{r}$ and $\boldsymbol{c}$ in order to simplify (13) below:

$$\tilde{N}_{m,n}(\boldsymbol{r},\boldsymbol{c}) = \binom{mn}{\sum_{k=1}^{n} c_k}^{-1} \prod_{i=1}^{m} \binom{n}{r_i} \prod_{j=1}^{n} \binom{m}{c_j} \exp\Big(-\frac{1}{2}\big(1-\mu_{m,n}(\boldsymbol{r},\boldsymbol{c})\big)\big(1-\nu_{m,n}(\boldsymbol{c})\big)\Big),$$

$$\mu_{m,n}(\boldsymbol{r},\boldsymbol{c}) = \eta_{m,n}(\boldsymbol{c}) \sum_{i=1}^{m} \Big(r_i - \frac{1}{m}\sum_{k=1}^{n} c_k\Big)^2, \qquad \nu_{m,n}(\boldsymbol{c}) = \eta_{m,n}(\boldsymbol{c}) \sum_{j=1}^{n} \Big(c_j - \frac{1}{n}\sum_{k=1}^{n} c_k\Big)^2,$$

$$\eta_{m,n}(\boldsymbol{c}) = \frac{mn}{(\sum_{k=1}^{n} c_k)(mn - \sum_{k=1}^{n} c_k)}.$$

Substituting this into (12) and simplifying gives

$$u_i = \frac{r_i}{n-r_i} \exp\Bigg[\eta_{m,n-1}(\boldsymbol{c}^{2:n})\big(1-\nu_{m,n-1}(\boldsymbol{c}^{2:n})\big)\Big(\frac{1}{2} - r_i + \frac{1}{m}\sum_{k=2}^{n} c_k\Big)\Bigg]. \tag{13}$$

If $r_i = 0$ or $r_i = n$, then the value of X_i is determined by $\tilde{\Omega}$, and any choice of $u_i > 0$ gives the same Q; we use $u_i = 1$ in these cases. We find that (13) works well over a large range of margins when P^* is uniform. It is excellent if the margins are approximately semi-regular, that is, if the row and column sums do not deviate substantially from their respective mean values.

For certain pathological cases with wildly varying margins, such as those in Bezáková et al. (2006), (13) does not work well. However, if the margins are such that the resulting matrices have a very low density of ones, even if the margins are highly irregular, then good performance can be obtained by instead using the asymptotic approximation of N from Greenhill et al. (2006, Theorem 1.3). Details are provided in the supplementary material. In fact, for the specific pathological cases in Bezáková et al. (2006) using this alternative approximation gives $Q^* \equiv P^*$ in the uniform case. None of the computationally efficient combinatorial approximations that we have found in literature work well when the margins are both highly irregular and lead to a moderate density of ones, but we are hopeful that advances in asymptotic enumeration techniques will eventually lead to approximations that work well in almost all cases.

Chen et al. (2005) observed that combinatorial approximations could be used to find a good choice of $\boldsymbol{u}$ and they mentioned an early asymptotic approximation from O'Neil (1969), which was explored further by Blanchet (2009) in an asymptotic analysis of the algorithm. The examples in Chen et al. (2005), however, use $u_i = r_i/(n-r_i)$, which is motivated by considering only the row margin constraints. Although there are several substantial differences between their proposal distribution and ours for the special case of the uniform distribution of Ω^*, we suspect that much of the improved performance of our algorithm results from using more accurate combinatorial approximations.

In the next section we use V and $\tilde{V}$ to account for the effects of $\boldsymbol{w}$, including the effects of zeros in $\boldsymbol{w}$. Since these zeros affect the size of the support of P^*, an alternative, perhaps more natural approach is to allow U and $\tilde{U}$ to capture the effects of zeros in $\boldsymbol{w}$. The supplementary material contains more details.

4.3 Non-uniform weighting

Here we discuss approximation of V by $\tilde{V}$. To develop an approximation, we will ignore the column margins and consider only the row margins. This is similar to the approach used by Chen et al. (2005) to develop combinatorial approximations for the uniform case. Let $\boldsymbol{B} \in \{0,1\}^{m\times n}$ be a matrix of independent Bernoulli(1/2) random variables, so that

$$\begin{aligned} V(\boldsymbol{x}) &= \mathbb{E}\Big(\prod_{ij} w_{ij}^{Y_{ij}} \Big| \boldsymbol{Y}^1 = \boldsymbol{x}\Big) = \mathbb{E}\Big(\prod_{ij} w_{ij}^{B_{ij}} \Big| \boldsymbol{B}^1 = \boldsymbol{x}, \boldsymbol{R}(\boldsymbol{B}) = \boldsymbol{r}, \boldsymbol{C}(\boldsymbol{B}) = \boldsymbol{c}\Big) \\ &\approx \mathbb{E}\Big(\prod_{ij} w_{ij}^{B_{ij}} \Big| \boldsymbol{B}^1 = \boldsymbol{x}, \boldsymbol{R}(\boldsymbol{B}) = \boldsymbol{r}\Big) = \prod_{i=1}^{m} \mathbb{E}\Big(\prod_{j=1}^{n} w_{ij}^{B_{ij}} \Big| B_{i1} = x_i, R_i(\boldsymbol{B}) = r_i\Big) \\ &\propto \prod_{i=1}^{m} v_i^{x_i}, \end{aligned} \tag{14}$$

where

$$\begin{aligned} v_i &= \frac{\mathbb{E}\big(\prod_{j=1}^{n} w_{ij}^{B_{ij}} \big| B_{i1} = 1, R_i(\boldsymbol{B}) = r_i\big)}{\mathbb{E}\big(\prod_{j=1}^{n} w_{ij}^{B_{ij}} \big| B_{i1} = 0, R_i(\boldsymbol{B}) = r_i\big)} \qquad (r_i = 1, \dots, n-1;\ i = 1, \dots, m) \\ &= \frac{w_{i1} \binom{n-1}{r_i - 1}^{-1} \sum_{\boldsymbol{b}\in\{0,1\}^{n-1}} \mathbb{1}\{\sum_{j=1}^{n-1} b_j = r_i - 1\} \prod_{j=2}^{n} w_{ij}^{b_{j-1}}}{\binom{n-1}{r_i}^{-1} \sum_{\boldsymbol{b}\in\{0,1\}^{n-1}} \mathbb{1}\{\sum_{j=1}^{n-1} b_j = r_i\} \prod_{j=2}^{n} w_{ij}^{b_{j-1}}}. \end{aligned} \tag{15}$$

In the supplementary material we describe how to compute all possible $\boldsymbol{v}$ for all columns using $O(nd)$ operations (where $d = \sum_i r_i = \sum_j c_j$) in a one-time preprocessing step that can be done prior to sampling. As with u_i, we always define $v_i = 1$ if $r_i = 0$ or $r_i = n$. For cases where $\boldsymbol{w}$ has zeros, we can sometimes have a zero in the denominator of (15) for $r_i > 0$. This happens when fewer than r_i of the $n-1$ remaining weights in the row are nonzero. Consequently, we need to force $X_i = 1$, which we do by setting the corresponding $\mathcal{A}_\ell = \{1\}$ in Section 4.1.

An important observation that we have thus far neglected is that many different choices of $\boldsymbol{w}$ give rise to the same P^*. Define

$$\Lambda(\boldsymbol{w}) = \{\boldsymbol{t} \in [0,\infty)^{m\times n} : \boldsymbol{t} = \boldsymbol{\alpha}\boldsymbol{\beta}^{\mathrm{T}} \circ \boldsymbol{w},\ \boldsymbol{\alpha} \in (0,\infty)^{m\times 1},\ \boldsymbol{\beta} \in (0,\infty)^{n\times 1}\},$$

where T denotes transpose and $\boldsymbol{y} \circ \boldsymbol{z}$ is the Hadamard product, that is, element-wise multiplication of matrices of the same size, defined by $(\boldsymbol{y} \circ \boldsymbol{z})_{ij} = y_{ij} z_{ij}$. Then for every $\boldsymbol{t} \in \Lambda(\boldsymbol{w})$ it is straightforward to verify that the P^* defined with the weight matrix $\boldsymbol{w}$ and the P^* defined with the weight matrix $\boldsymbol{t}$ are identical. Similarly, the two versions of V differ only by an inconsequential constant of proportionality. Unfortunately, our approximation $\tilde{V}(\boldsymbol{x}) \propto \prod_i v_i^{x_i}$ defined above does not share this invariance. Consequently, proposal distributions constructed with $\boldsymbol{w}$ and $\boldsymbol{t}$, respectively, could differ, even though the target distribution does not differ. We find this unappealing and remedy it in a preprocessing step prior to the construction of Q^* by first transforming $\boldsymbol{w}$ into an equivalent, canonical $\bar{\boldsymbol{w}} \in \Lambda(\boldsymbol{w})$. In particular, $\bar{\boldsymbol{w}}$ is the unique element of $\Lambda(\boldsymbol{w})$ with the property that its average nonzero entry over any row or column is one, namely,

$$\sum_{i=1}^{m} \bar{w}_{ij} = \sum_{i=1}^{m} \mathbb{1}\{\bar{w}_{ij} > 0\}, \quad \sum_{j=1}^{n} \bar{w}_{ij} = \sum_{j=1}^{n} \mathbb{1}\{\bar{w}_{ij} > 0\} \quad (i = 1, \dots, m;\ j = 1, \dots, n). \tag{16}$$

The solution to (16) over $\Lambda(\boldsymbol{w})$ exists, is unique, and is easy to find numerically (Rothblum & Schneider, 1989); see supplementary material for details and more discussion. In the examples below, we always define $\boldsymbol{v}$ in terms of $\bar{\boldsymbol{w}}$, not $\boldsymbol{w}$. Not only does this ensure that Q^* has the same invariance property as P^*, but we find that performance of the algorithm tends to improve.

If we know that P^* is uniform over Ω^*, for example, if $\boldsymbol{w} \equiv 1$ or $\bar{\boldsymbol{w}} \equiv 1$, then V is constant over Ω, and we can ignore $\boldsymbol{v}$ in the construction of Q.

5 Importance sampling

5.1 Algorithm summary

1. Preprocessing: Compute $\bar{\boldsymbol{w}}$ from (16) and precompute all possible $\boldsymbol{v}$ using (15) with $\bar{\boldsymbol{w}}$ in place of $\boldsymbol{w}$; see supplementary material for details. Compute $\boldsymbol{\pi}$, $\boldsymbol{\mathcal{A}}$, and $\boldsymbol{\mathcal{B}}$ according to Theorem 2 and $\boldsymbol{u}$ according to (13) for the first column.

2. Generating a single observation, $\tilde{\boldsymbol{Z}}$, from Q^*: The matrix $\tilde{\boldsymbol{Z}}$ is generated column-by-column as follows. Set $q = 1$. Sequentially, for each column:

   (a) For the current $m, n, \boldsymbol{r}, \boldsymbol{c}$, compute $\boldsymbol{\pi}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}, \boldsymbol{u}$ as above. After the preprocessing, updating these quantities based on the previously sampled column requires $O(m)$ operations.

   (b) Use Theorem 1 to compute the Markov chain representation for Q. For the jth column, this takes $O(mc_j)$ operations. If $Q \equiv 0$, then $\tilde{\boldsymbol{Z}}$ will not be in the support of P^*; assign a final importance weight of zero and go to step 3.

   (c) Generate a random observation $\boldsymbol{X}$ from Q and evaluate $Q(\boldsymbol{X})$. This takes $O(m)$ operations.

   (d) Assign the current column of $\tilde{\boldsymbol{Z}}$ to be $\boldsymbol{X}$. Update $q \leftarrow qQ(\boldsymbol{X})$, $\boldsymbol{c} \leftarrow \boldsymbol{c}^{2:n}$, $n \leftarrow n - 1$, $\boldsymbol{r} \leftarrow \boldsymbol{r} - \boldsymbol{X}$. If $n > 0$, continue looping over columns; go to step 2a. If $n = 0$, the final matrix is $\tilde{\boldsymbol{Z}}$, and we have $Q^*(\tilde{\boldsymbol{Z}}) = q$; go to step 3.

3. To generate additional independent observations from Q^*, reset all variables to their original values after step 1 and repeat step 2.

The same algorithm can be used to evaluate $Q^*(\boldsymbol{z})$ for any $\boldsymbol{z} \in \Omega^*$. Simply assign $\boldsymbol{X}$ to be the current column of $\boldsymbol{z}$ in step 2c, instead of sampling a new column. (The algorithm can be applied for any ordering of the columns, and Q^* will depend on the chosen ordering. The supplementary material describes the heuristics that we use to choose a column ordering.)

5.2 Monte Carlo approximation and diagnostics

Let $\boldsymbol{Z}$ have distribution P^*, let $\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_T$ be random sample from Q^* generated as above, and let h be a function over Ω^*. Define the unnormalized importance weights

$$f(\boldsymbol{z}) = \frac{\kappa P^*(\boldsymbol{z})}{Q^*(\boldsymbol{z})} = \frac{\prod_{ij} w_{ij}^{z_{ij}} \mathbb{1}\{\boldsymbol{z} \in \Omega^*\}}{Q^*(\boldsymbol{z})} \qquad (\boldsymbol{z} \in \{0,1\}^{m \times n}),$$

which we can efficiently evaluate for any $\boldsymbol{z}$ as described above. In the formula for $f(\boldsymbol{z})$ it is important that we use $\boldsymbol{w}$, not $\bar{\boldsymbol{w}}$, particularly if we are approximating κ. We can

approximate κ and $\mu = \mathbb{E}(h(\boldsymbol{Z}))$ via importance sampling in the usual way, namely,

$$\hat{\mu}_T = \frac{\sum_{t=1}^T f(\boldsymbol{Z}_t)h(\boldsymbol{Z}_t)}{\sum_{t=1}^T f(\boldsymbol{Z}_t)} \to \mu, \qquad \hat{\kappa}_T = \frac{1}{T}\sum_{t=1}^T f(\boldsymbol{Z}_t) \to \kappa \qquad (T \to \infty). \tag{17}$$

Besides being consistent, $\hat{\kappa}_T$ and $\hat{\kappa}_T\hat{\mu}_T$ are also unbiased approximations of κ and $\kappa\mu$, respectively. See Liu (2001) for details about importance sampling. See Harrison (2012) for modifications when (17) is used to approximate a p-value.

In this context, importance sampling algorithms are usually evaluated empirically by diagnostics related to the variability of the importance weights. The less variable the importance weights, the better the algorithm is judged to be performing. For the numerical illustrations below, we report

$$\hat{\text{cv}}_T^2 = \frac{1}{(T-1)\hat{\kappa}_T^2}\sum_{t=1}^T (f(\boldsymbol{Z}_t) - \hat{\kappa}_T)^2, \qquad \hat{\Delta}_T = \frac{\max_{t=1,\dots,T} f(\boldsymbol{Z}_t)}{\min_{t=1,\dots,T} f(\boldsymbol{Z}_t)} - 1.$$

As $T \to \infty$, the approximate squared coefficient of variation, $\hat{\text{cv}}_T^2$, converges to the true squared coefficient of variation, $\text{cv}^2 = \mathbb{V}(f(\boldsymbol{Z}_1))/\,\mathbb{E}(f(\boldsymbol{Z}_1))^2 = \mathbb{V}(P^*(\boldsymbol{Z}_1)/Q^*(\boldsymbol{Z}_1))$, where $\mathbb{V}$ denotes variance. $\tilde{T} = T/(1+\text{cv}^2)$ has been suggested as a rough diagnostic for effective sample size, meaning that a sample size of T from Q^* behaves roughly like a sample size of $\tilde{T}$ from P^* for the purposes of Monte Carlo approximating μ for well-behaved functions h (Kong et al., 1994; Liu, 2001). For many but not all examples we find $\hat{\text{cv}}_T^2 < 1$, suggesting that Q^* is appropriate for efficient importance sampling. The relative range of importance weights reported by $\hat{\Delta}_T$ is an especially stringent diagnostic. For nearly constant margins and P^* close to uniform, we often find $\hat{\Delta}_T \approx 0$, suggesting that Q^* is an excellent approximation of P^*; see Table 1.

6 Numerical illustrations

We experiment with four different classes of weights based on a canonical matrix $\boldsymbol{y}$ whose entries are independently sampled from the uniform$(0,1)$ distribution: (I) $w_{ij} = 1$, which is the uniform distribution over Ω^*, (II) $w_{ij} = y_{ij} + 1$, (III) $w_{ij} = y_{ij}$, and (IV) $w_{ij} = -\mathbb{1}(y_{ij} < 0.99)\log(y_{ij})$, for all i, j. The specific entries of $\boldsymbol{y}$ for different sized matrices are in the supplementary material. The resulting P^* is increasingly non-uniform in each of the latter three cases and has 1% structural zeros in case (IV). Recall that each $\boldsymbol{w}$ corresponds to a family of weights of the form $\boldsymbol{\alpha}\boldsymbol{\beta}^{\mathrm{T}} \circ \boldsymbol{w}$ that give the same P^* and Q^*; see Section 4.3. In all cases we report results with $T = 1000$.

We begin with 500×500 r_1-regular matrices, i.e., $r_i = c_j = r_1$ for all $i, j = 1, \dots, 500$. Results are summarized in Table 1 for $r_1 = 1, 2, 4, 8, \dots, 256$. The diagnostics are striking, especially in the uniform case, for which the importance weights are essentially constant. Performance degrades slightly as P^* becomes strongly non-uniform, but in all cases the estimated cv^2 is less than one. Low variability in importance weights corresponds to high precision in estimates of κ. For example, in the uniform case, where $\kappa = |\Omega^*|$, for $r_1 = 256$ we obtain $\hat{\kappa} = (1.478301 \pm 0.000044) \times 10^{73781}$, where the errors are approximate standard errors estimated from the same importance samples, and for $r_1 = 2$ we obtain $\hat{\kappa}_T = (2.27653 \pm 0.00017) \times 10^{2266}$, the latter of which is close to the true value of $\kappa = 2.27658\ldots \times 10^{2266}$; see

supplementary material. To our knowledge the exact value of κ in these examples can only be efficiently computed for the special case of the uniform distribution over either 1-regular or 2-regular matrices (Anand et al., 1966). Sampling from the uniform distribution over 1-regular matrices is trivial, $\kappa = m!$, and there is no need to use our algorithm, although it is comforting that $Q^* = P^*$ in this case.

We remark that the distributions corresponding to different weight classes in Table 1 are almost singular with respect to each other. For example, in the 1-regular case, if we use the Q^* for weight class I as a proposal distribution for the P^* corresponding to one of the other weight classes, then we obtain, for weight class II, $\hat{\Delta}_T = 2{\times}10^{13}$ and $\hat{\mathrm{cv}}_T^2 = 8{\times}10^2$, for weight class III, $\hat{\Delta}_T = 2{\times}10^{67}$ and $\hat{\mathrm{cv}}_T^2 = 1{\times}10^3$, and for weight class IV, only 6 of the 1000 observations from Q^* were even in the support of P^*, owing to the structural zeros. Results are similar for other combinations and become even more extreme as r_1 increases.

Table 1: 500×500 r_1-regular matrices

	uniform		$\boldsymbol{w}$ class II		$\boldsymbol{w}$ class III		$\boldsymbol{w}$ class IV	
r_1	$\hat{\Delta}_T$	$\hat{\mathrm{cv}}_T^2$	$\hat{\Delta}_T$	$\hat{\mathrm{cv}}_T^2$	$\hat{\Delta}_T$	$\hat{\mathrm{cv}}_T^2$	$\hat{\Delta}_T$	$\hat{\mathrm{cv}}_T^2$
1	0	0	$2{\times}10^{-1}$	$5{\times}10^{-4}$	$4{\times}10^{0}$	$4{\times}10^{-2}$	$5{\times}10^{1}$	$3{\times}10^{-1}$
2	$4{\times}10^{-2}$	$5{\times}10^{-6}$	$2{\times}10^{-1}$	$4{\times}10^{-4}$	$6{\times}10^{0}$	$4{\times}10^{-2}$	$8{\times}10^{1}$	$2{\times}10^{-1}$
4	$1{\times}10^{-2}$	$1{\times}10^{-6}$	$1{\times}10^{-1}$	$4{\times}10^{-4}$	$5{\times}10^{0}$	$3{\times}10^{-2}$	$2{\times}10^{2}$	$2{\times}10^{-1}$
8	$2{\times}10^{-2}$	$1{\times}10^{-6}$	$2{\times}10^{-1}$	$3{\times}10^{-4}$	$3{\times}10^{0}$	$3{\times}10^{-2}$	$4{\times}10^{1}$	$2{\times}10^{-1}$
16	$1{\times}10^{-2}$	$1{\times}10^{-6}$	$2{\times}10^{-1}$	$3{\times}10^{-4}$	$3{\times}10^{0}$	$3{\times}10^{-2}$	$4{\times}10^{1}$	$1{\times}10^{-1}$
32	$8{\times}10^{-3}$	$8{\times}10^{-7}$	$1{\times}10^{-1}$	$2{\times}10^{-4}$	$2{\times}10^{0}$	$2{\times}10^{-2}$	$1{\times}10^{1}$	$1{\times}10^{-1}$
64	$9{\times}10^{-3}$	$9{\times}10^{-7}$	$1{\times}10^{-1}$	$2{\times}10^{-4}$	$3{\times}10^{0}$	$2{\times}10^{-2}$	$2{\times}10^{1}$	$9{\times}10^{-2}$
128	$1{\times}10^{-2}$	$9{\times}10^{-7}$	$1{\times}10^{-1}$	$9{\times}10^{-5}$	$1{\times}10^{0}$	$1{\times}10^{-2}$	$5{\times}10^{0}$	$5{\times}10^{-2}$
256	$9{\times}10^{-3}$	$9{\times}10^{-7}$	$5{\times}10^{-2}$	$5{\times}10^{-5}$	$1{\times}10^{0}$	$1{\times}10^{-2}$	$9{\times}10^{0}$	$7{\times}10^{-2}$

For the special case of 1-regular matrices, corresponding to the first row in Table 1, κ is the permanent of $\boldsymbol{w}$ and various generalizations of the permanent correspond to expectations under P^*. The current state-of-the-art algorithm for approximating permanents and α-permanents of general matrices, see (3) above, seems to be the importance sampling algorithm of Kou & McCullagh (2009), which has about the same computational complexity as our algorithm. For the case $\alpha = 1$, their algorithm is nearly identical to ours, the main differences being the choice of column order and our use of $\bar{\boldsymbol{w}}$, which seems to give our algorithm slightly better performance. Their algorithm is generally preferable for $\alpha \neq 1$, since it is tailored to the specific choice of α, although in many cases performance is comparable. The supplementary materials have numerical comparisons for each of the $\boldsymbol{w}$ used in Table 1 and for all of the examples in Kou & McCullagh (2009), which include cases with $\alpha = 1$ and $\alpha = 1/2$. It is interesting that in many cases our generic approach is competitive with specialized software.

In Table 2 we repeat the simulations of Table 1 for 50×100 irregular matrices with margins $\boldsymbol{r} = k\tilde{\boldsymbol{r}}$ and $\boldsymbol{c} = k\tilde{\boldsymbol{c}}$, for the cases $k = 1, \ldots, 4$, where $\tilde{\boldsymbol{r}}^{\mathrm{T}} = (24^1,\ 22^2,\ 17^4,\ 13^3,\ 12^2,\ 11^3,\ 10^2,\ 9^3,\ 8^6,\ 7^1,\ 6^4,\ 5^4,\ 4^5,\ 3^6,\ 2^4)$ and $\tilde{\boldsymbol{c}} = (12^2,\ 10^2,\ 9^5,\ 8^4,\ 7^6,\ 6^{11},\ 5^{10},\ 4^{18},\ 3^9,\ 2^{13},\ 1^{20})$ using i^j to denote j copies of i. Performance degrades in the irregular case as the matrices become more dense. In most, but not all cases, the diagnostics suggest Q^* could be used for efficient importance sampling.

For the special case of the uniform distribution, corresponding to the far left category of weights in Tables 1 and 2, the sequential importance sampling algorithm of Chen et al. (2005), as implemented in the publicly available R package `networksis` (Admiraal & Handcock,

Table 2: 50×100 irregular matrices with $\boldsymbol{r} = k\tilde{\boldsymbol{r}}$, $\boldsymbol{c} = k\tilde{\boldsymbol{c}}$

	uniform		$\boldsymbol{w}$ class II		$\boldsymbol{w}$ class III		$\boldsymbol{w}$ class IV	
k	$\hat{\Delta}_T$	$\hat{\text{CV}}_T^2$	$\hat{\Delta}_T$	$\hat{\text{CV}}_T^2$	$\hat{\Delta}_T$	$\hat{\text{CV}}_T^2$	$\hat{\Delta}_T$	$\hat{\text{CV}}_T^2$
1	4×10^{-1}	1×10^{-3}	3×10^{0}	5×10^{-2}	8×10^{1}	5×10^{-1}	5×10^{3}	3×10^{0}
2	3×10^{0}	3×10^{-2}	7×10^{0}	1×10^{-1}	7×10^{2}	2×10^{0}	6×10^{4}	7×10^{0}
3	2×10^{2}	7×10^{-1}	2×10^{2}	6×10^{-1}	2×10^{4}	6×10^{0}	3×10^{6}	4×10^{1}
4	3×10^{6}	3×10^{1}	3×10^{6}	2×10^{1}	4×10^{9}	2×10^{2}	2×10^{13}	8×10^{2}

2008), appears to be the current state-of-the-art algorithm for practical Monte Carlo approximation. Our algorithm is a substantial improvement, especially for dense or irregular margins. Using `networksis` gives $\hat{\Delta}_T = (1\times10^{1}, 1\times10^{2}, 2\times10^{5}, 1\times10^{11})$ and $\hat{\text{CV}}_T^2 = (2\times10^{-1}, 6\times10^{-1}, 1\times10^{1}, 4\times10^{2})$ for the first pair of columns in Table 2. The `networksis` implementation is several orders of magnitude slower than our implementation, and is too slow for most of the examples in Table 1.

Supplementary Material

Supplementary material includes (i) a more detailed description of the column-wise factorization described in Section 3.1, (ii) details about the solution to (16) and other preprocessing of the weights and margins, (iii) alternative combinatorial approximations for sparse matrices with irregular margins, (iv) extensions to Theorem 2 for the case of structural zeros with at most one structural zero in each row and column, including the case of structural zeros along the diagonal, (v) more principled treatments of structural zeros in the approximations to U and V, (vi) details for the numerical simulations, (vii) additional numerical illustrations, including examples using real data, and (viii) a Matlab implementation of the algorithm.

A Column-wise factorization

Define the set of binary matrices with margins $\boldsymbol{r} \in \mathbb{N}^{m\times 1}$ and $\boldsymbol{c} \in \mathbb{N}^{1\times n}$ to be

$$\Omega^*_{m,n}(\boldsymbol{r}, \boldsymbol{c}) = \{\boldsymbol{z} \in \{0,1\}^{m\times n} : \boldsymbol{R}(\boldsymbol{z}) = \boldsymbol{r}, \boldsymbol{C}(\boldsymbol{z}) = \boldsymbol{c}\},$$

and let

$$N_{m,n}(\boldsymbol{r}, \boldsymbol{c}) = |\Omega^*_{m,n}(\boldsymbol{r}, \boldsymbol{c})|$$

denote the number of such matrices, where $\mathbb{N} = \{0, 1, \dots\}$ denotes the nonnegative integers. For an $m \times n$ matrix $\boldsymbol{w} \in [0, \infty)^{m\times n}$ define the function

$$P^*_{m,n}(\boldsymbol{z} \mid \boldsymbol{r}, \boldsymbol{c}, \boldsymbol{w}) = \frac{\mathbb{1}\{\boldsymbol{z} \in \Omega^*_{m,n}(\boldsymbol{r}, \boldsymbol{c})\}}{\kappa_{m,n}(\boldsymbol{r}, \boldsymbol{c}, \boldsymbol{w})} \prod_{i=1}^{m}\prod_{j=1}^{n} w_{ij}^{z_{ij}} \qquad (\boldsymbol{z} \in \{0,1\}^{m\times n})$$

with normalization constant

$$\kappa_{m,n}(\boldsymbol{r}, \boldsymbol{c}, \boldsymbol{w}) = \sum_{\boldsymbol{z}\in\Omega^*_{m,n}(\boldsymbol{r},\boldsymbol{c})} \prod_{i=1}^{m}\prod_{j=1}^{n} w_{ij}^{z_{ij}}$$

using the convention that $0/0 = 0$. If $\kappa_{m,n}(\boldsymbol{r}, \boldsymbol{c}, \boldsymbol{w}) > 0$, then $P^*_{m,n}$ is a probability mass function and we use $\boldsymbol{Z}$ to denote a random binary matrix with this distribution. We use $\boldsymbol{Z}^1$ to denote the first column of $\boldsymbol{Z}$, which has probability mass function

$$P_{m,n}(\boldsymbol{x} \mid \boldsymbol{r}, \boldsymbol{c}, \boldsymbol{w}) = \mathbb{P}(\boldsymbol{Z}^1 = \boldsymbol{x}) = \sum_{\boldsymbol{z} \in \Omega^*_{m,n}(\boldsymbol{r}, \boldsymbol{c})} P^*_{m,n}(\boldsymbol{z} \mid \boldsymbol{r}, \boldsymbol{c}, \boldsymbol{w}) \mathbb{1}\{\boldsymbol{z}^1 = \boldsymbol{x}\} \qquad (\boldsymbol{x} \in \{0,1\}^{m \times 1}),$$

the support of which is a subset of

$$\Omega_{m,n}(\boldsymbol{r}, \boldsymbol{c}) = \{\boldsymbol{x} \in \{0,1\}^{m \times 1} : \boldsymbol{x} = \boldsymbol{z}^1,\ \boldsymbol{z} \in \Omega^*_{m,n}(\boldsymbol{r}, \boldsymbol{c})\}.$$

If the entries of $\boldsymbol{w}$ are strictly positive, then the support is all of $\Omega_{m,n}(\boldsymbol{r}, \boldsymbol{c})$.

The algorithmic challenge of sampling the entire matrix $\boldsymbol{Z}$ reduces to the challenge of sampling from the first column $\boldsymbol{Z}^1$, because once we have the first column, then we can update the margins and proceed sequentially, treating successive columns like the first. Indeed, it is straightforward to verify that

$$\mathbb{P}(\boldsymbol{Z}^j = \boldsymbol{x} \mid \boldsymbol{Z}^{1:j-1} = \boldsymbol{y}) = P_{m,n-j+1}(\boldsymbol{x} \mid \boldsymbol{r} - \boldsymbol{R}(\boldsymbol{y}), \boldsymbol{c}^{j:n}, \boldsymbol{w}^{j:n}),$$

so that the generic decomposition $\mathbb{P}(\boldsymbol{Z}) = \mathbb{P}(\boldsymbol{Z}^1) \prod_{j=2}^n \mathbb{P}(\boldsymbol{Z}^j \mid \boldsymbol{Z}^{1:j-1})$ gives

$$P^*_{m,n}(\boldsymbol{z} \mid \boldsymbol{r}, \boldsymbol{c}, \boldsymbol{w}) = P_{m,n}(\boldsymbol{z}^1 \mid \boldsymbol{r}, \boldsymbol{c}, \boldsymbol{w}) \prod_{j=2}^{n} P_{m,n-j+1}(\boldsymbol{z}^j \mid \boldsymbol{r} - \boldsymbol{R}(\boldsymbol{z}^{1:j-1}), \boldsymbol{c}^{j:n}, \boldsymbol{w}^{j:n}).$$

(In the main text, we primarily focus on sampling the first column $\boldsymbol{Z}_1$, we suppress $m, n, \boldsymbol{r}, \boldsymbol{c}, \boldsymbol{w}$ in the notation as much as possible, and we assume that $\kappa > 0$.) To summarize, our target distribution is the binary random vector $\boldsymbol{Z}_1 \in \{0,1\}^{m \times 1}$ with probability mass function

$$P(\boldsymbol{x}) \propto \sum_{\boldsymbol{z} \in \{0,1\}^{m \times n}} \mathbb{1}\{\boldsymbol{R}(\boldsymbol{z}) = \boldsymbol{r}, \boldsymbol{C}(\boldsymbol{z}) = \boldsymbol{c}, \boldsymbol{z}^1 = \boldsymbol{x}\} \prod_{i=1}^{m} \prod_{j=1}^{n} w_{ij}^{z_{ij}}.$$

B Preprocessing the weights and margins

The preprocessing that affects the definition of Q^* consists of transforming $\boldsymbol{w}$ into $\bar{\boldsymbol{w}}$ and choosing an ordering of the columns. Other elements of the preprocessing are merely for computational efficiency. All of the preprocessing of $\boldsymbol{w}$, but not reordering the columns, can be skipped when it is known that P^* is uniform over Ω^*, e.g., when $\bar{\boldsymbol{w}} \equiv 1$.

B.1 Computing $\bar{w}$, the solution to equation (16) in the main text

Fix $\boldsymbol{w} \in [0, \infty)^{m \times n}$. Define

$$n_i = \sum_{j=1}^{n} \mathbb{1}\{w_{ij} > 0\}, \quad m_j = \sum_{i=1}^{m} \mathbb{1}\{w_{ij} > 0\} \quad (i = 1, \ldots, m;\ j = 1, \ldots, n).$$

We are looking for the $\bar{\boldsymbol{w}} \in [0, \infty)^{m \times n}$ with the following properties:

$$\bar{w}_{ij} = \alpha_i \beta_j w_{ij}, \quad \sum_{j=1}^{n} \bar{w}_{ij} = n_i, \quad \sum_{i=1}^{m} \bar{w}_{ij} = m_j \quad (\alpha_i, \beta_j > 0;\ i = 1, \ldots, m;\ j = 1, \ldots, n).$$

Initializing $\bar{\boldsymbol{w}}^{(0)} = \boldsymbol{w}$ and $t = 1$, we iterate the following fixed point equations until convergence:

$$\bar{w}_{ij}^{(2t-1)} = \frac{n_i \bar{w}_{ij}^{(2t-2)}}{\sum_{\ell=1}^{n} \bar{w}_{i\ell}^{(2t-2)}} \quad (i = 1, \ldots, m;\ j = 1, \ldots, n)$$

$$\bar{w}_{ij}^{(2t)} = \frac{m_j \bar{w}_{ij}^{(2t-1)}}{\sum_{\ell=1}^{m} \bar{w}_{\ell j}^{(2t-1)}} \quad (i = 1, \ldots, m;\ j = 1, \ldots, n),$$

where superscripts are indices, and where we take $0/0 = 0$. If we iterate this for T steps, then we use $\bar{\boldsymbol{w}} = \bar{\boldsymbol{w}}^{(2T)}$. In our experience, a small T is usually adequate to reach convergence. Since each $\bar{\boldsymbol{w}}^{(T)} \in \Lambda(\boldsymbol{w})$, iterating to convergence is not important for validity of the algorithm. Rothblum & Schneider (1989) prove existence and uniqueness of $\bar{\boldsymbol{w}}$. They also show that the solution can also be found using a convex programming algorithm, but we have not experimented with this approach.

The computational cost of computing $\bar{\boldsymbol{w}}$ takes at least $O(mn)$ operations, but we do not have theoretical bounds on the computational complexity. In our experience, it can be treated as a negligible preprocessing step. Although this choice of $\bar{\boldsymbol{w}}$ outperforms many alternatives, we have found no theoretical justification for its use. It is closely related to Sinkhorn balancing of $\boldsymbol{w}$ (Sinkhorn, 1964, 1967), which has appeared in the literature in both algorithmic and theoretical treatments of permanents (e.g., Ando, 1989; Beichl & Sullivan, 1999), and it has the nice property that $\bar{\boldsymbol{w}} \equiv 1$ whenever $\boldsymbol{w} = \boldsymbol{\alpha}\boldsymbol{\beta}^{\mathrm{T}}$. In any case, $P^*_{m,n}(\cdot \mid \boldsymbol{r}, \boldsymbol{c}, \bar{\boldsymbol{w}}) = P^*_{m,n}(\cdot \mid \boldsymbol{r}, \boldsymbol{c}, \boldsymbol{w})$, so switching from $\boldsymbol{w}$ to $\bar{\boldsymbol{w}}$ does not change the target distribution.

B.2 Choosing a column ordering

Our algorithm is not invariant to the ordering of the columns, nor to the pattern of zeros in $\bar{\boldsymbol{w}}$. We use the following heuristic ordering of the columns. First, if $\bar{\boldsymbol{w}}$ is banded, then we leave the columns in their original order. The special case of banded weights arises frequently in some applications and we find that the banded ordering works best for accommodating so many zero weights. In other cases, we reorder the columns first by decreasing column sum and then by decreasing variance of the entries of $\bar{\boldsymbol{w}}$ within each column. These preprocessing steps, and the accompanying postprocessing steps of returning the columns to their original orders, all require negligible additional computation. In practice, if one is interested in a specific matrix for which these heuristics do not work well, then it can often be advantageous to experiment with different column orders or perhaps swapping the roles of rows and columns. For the description of the algorithm, when referring to the jth column, we mean the jth column after any reordering of the columns.

B.3 Precomputing the constants v for all columns

Define the symmetric polynomials

$$G_n(\boldsymbol{y}, k) = \sum_{\boldsymbol{b} \in \{0,1\}^n} \mathbb{1}\{\textstyle\sum_{j=1}^{n} b_j = k\} \displaystyle\prod_{j=1}^{n} y_j^{b_j} \qquad (\boldsymbol{y} \in \mathbb{R}^n,\ k = 0, \ldots, n), \tag{18}$$

and let $\bar{\boldsymbol{w}}_i^{j:k} = (\bar{w}_{ij}, \ldots, \bar{w}_{ik})$ denote the ith row of $\bar{\boldsymbol{w}}^{j:k}$. Before sampling we also precompute and store

$$G(i,j,k) = G_{n-j+1}(\bar{\boldsymbol{w}}_i^{j:n}, k) \tag{19}$$

for all $i = 1, \ldots, m$, $j = 1, \ldots, n$, and $k = 0, \ldots, \min(r_i, n-j+1)$. The entire collection can be computed in $O(nd)$ operations (where $d = \sum_i r_i = \sum_j c_j$), by initializing with $G(i,j,0) = 1$, $G(i,n,1) = \bar{w}_{in}$, and $G(i,j,k) = 0$ for all i,j and $k > n-j+1$, and then using the recursive formula

$$G(i,j,k) = G(i,j+1,k) + \bar{w}_{ij} G(i,j+1,k-1).$$

In particular, in equation (15) in the main text we see that for the first column

$$v_i = \frac{\bar{w}_{i1} \binom{n-1}{r_i-1}^{-1} G(i,2,r_i-1)}{\binom{n-1}{r_i}^{-1} G(i,2,r_i)}.$$

(Recall that we use $\bar{\boldsymbol{w}}$, not $\boldsymbol{w}$, in our implementation of the algorithm.) For the jth column $(1 < j < n)$ we will have

$$v_i = \frac{\bar{w}_{ij} \binom{n-j}{r_i - R_i(\boldsymbol{z}^{1:j-1}) - 1}^{-1} G(i,j+1,r_i - R_i(\boldsymbol{z}^{1:j-1}) - 1)}{\binom{n-j}{r_i - R_i(\boldsymbol{z}^{1:j-1})}^{-1} G(i,j+1,r_i - R_i(\boldsymbol{z}^{1:j-1}))},$$

where $\boldsymbol{z}^{1:j-1}$ are the previously sampled columns so that $\boldsymbol{r} - \boldsymbol{R}(\boldsymbol{z}^{1:j-1})$ are the updated row sums when preparing to sample the jth column.

C Alternative combinatorial approximations

For each positive integer ℓ and any nonnegative integer a we define

$$[a]_\ell = a(a-1)\cdots(a-\ell+1),$$

and for a k-vector $\boldsymbol{t}$ of nonnegative integers we define

$$[\boldsymbol{t}]_\ell = \sum_{i=1}^{k} [t_i]_\ell.$$

In Section 4.2 of the main text we used a combinatorial approximation due to Canfield et al. (2008), however, other approximations can also be used and may give better performance for some problems. For instance, Greenhill et al. (2006, Theorem 1.3) provide an alternative combinatorial approximation for $N_{m,n}(\boldsymbol{r},\boldsymbol{c})$ that is accurate, asymptotically, for sparse matrices, except perhaps when the margins are extremely variable:

$$\tilde{N}_{m,n}(\boldsymbol{r},\boldsymbol{c}) = \frac{[\boldsymbol{c}]_1!}{\prod_{i=1}^m r_i! \prod_{j=1}^n c_j!} \exp\bigl(-\alpha_1(\boldsymbol{c})[\boldsymbol{r}]_2 - \alpha_2(\boldsymbol{c})[\boldsymbol{r}]_3 - \alpha_3(\boldsymbol{c})[\boldsymbol{r}]_2^2\bigr),$$

$$\alpha_1(\boldsymbol{c}) = \frac{[\boldsymbol{c}]_2}{2[\boldsymbol{c}]_1^2} + \frac{[\boldsymbol{c}]_2}{2[\boldsymbol{c}]_1^3} + \frac{[\boldsymbol{c}]_2^2}{4[\boldsymbol{c}]_1^4}, \quad \alpha_2(\boldsymbol{c}) = -\frac{[\boldsymbol{c}]_3}{3[\boldsymbol{c}]_1^3} + \frac{[\boldsymbol{c}]_2^2}{2[\boldsymbol{c}]_1^4}, \quad \alpha_3(\boldsymbol{c}) = \frac{[\boldsymbol{c}]_2}{4[\boldsymbol{c}]_1^4} + \frac{[\boldsymbol{c}]_3}{2[\boldsymbol{c}]_1^4} - \frac{[\boldsymbol{c}]_2^2}{2[\boldsymbol{c}]_1^5},$$

where we take $0/0 = 0$. Following Section 4.2 in the main text, a straightforward calculation gives

$$u_i = r_i \exp\big[(r_i - 1)\big(2\alpha_1(\boldsymbol{c}^{2:n}) + 3\alpha_2(\boldsymbol{c}^{2:n})(r_i - 2) + 4\alpha_3(\boldsymbol{c}^{2:n})([\boldsymbol{r}]_2 - r_i + 1)\big)\big].$$

This combinatorial approximation is an improvement of an approximation in O'Neil (1969), which was mentioned in Chen et al. (2005) and studied by Blanchet (2009). Both are exactly uniform for the pathological cases in Bezáková et al. (2006); see Supplementary Section E.5.

D Structural zeros and ones

D.1 Remarks

The algorithm can be improved to better accommodate structural zeros. We avoided this in the main text to simplify the exposition, but the complexity of the algorithm does not change significantly. The numerical experiments in the main text do not use these improvements, even though some of the examples have structural zeros.

We use the term structural zeros to denote positions (i, j) such that $w_{ij} = 0$, which allows the investigator to explicitly force the binary matrix to zeros at those positions. It is possible that the row and column sums also force some entries to be zero, but we are not referring to those types of implicit structural zeros.

Sometimes it is desirable to force an entry to be one. These structural ones can be accommodated using structural zeros. In a preprocessing step, we replace structural ones with structural zeros and decrement the row and column sums appropriately. Then we sample as usual. In a postprocessing step, we reinsert the structural ones. Henceforth, we only discuss structural zeros.

D.2 Extensions to Theorem 2 in the main text for zero diagonal

Here we report an extension of Theorem 2 in the main text to the case where w has at most one zero entry in each row and column. This includes the special case of a zero diagonal, which arises frequently when the binary matrices of interest are adjacency matrices of directed graphs. Unlike the main text, the order of the columns is important for the validity of the algorithm. The columns must be reordered during preprocessing so that $c_1 \geq \cdots \geq c_n$.

Theorem 3. *(Chen, 2006, 2007) Assume that $c_1 \geq \cdots \geq c_n$, fix $\boldsymbol{w} \in [0, \infty)^{m \times n}$, define $a_{ij} = \mathbb{1}\{w_{ij} > 0\}$ for each i, j, assume $R_i(\boldsymbol{a}) \geq n - 1$ and $C_j(\boldsymbol{a}) \geq m - 1$ for each i, j, and assume that $\kappa_{m,n}(\boldsymbol{r}, \boldsymbol{c}, \boldsymbol{w}) > 0$. Choose $\boldsymbol{\pi}$ so that $r_{\pi_1} \geq \cdots \geq r_{\pi_m}$ and so that whenever $r_{\pi_i} = r_{\pi_{i+1}}$ we also have $y_{\pi_i} \leq y_{\pi_{i+1}}$, where*

$$y_i = \begin{cases} \text{the unique } j \text{ such that } w_{ij} = 0 & \text{if there exists such a } j; \\ n + 1 & \text{otherwise.} \end{cases}$$

For each $i = 1, \ldots, m$, define

$$\mathcal{A}_i = \begin{cases} \{0\} & (a_{\pi_i 1} r_{\pi_i} = 0); \\ \{0, 1\} & (0 < a_{\pi_i 1} r_{\pi_i} < R_{\pi_i}(a)); \\ \{1\} & (a_{\pi_i 1} r_{\pi_i} = R_{\pi_i}(a)), \end{cases} \qquad \mathcal{B}_i = \begin{cases} \{\max(0, b_i), \ldots, c_1\} & (i < m); \\ \{c_1\} & (i = m), \end{cases}$$

for

$$b_i = (\textstyle\sum_{\ell=1}^{i} r_{\pi_\ell}) - \min_{j=1,\dots,n}\{\sum_{k=j+1}^{n} c_k + \sum_{\ell=1}^{i}\sum_{k=2}^{j} a_{\pi_\ell k}\}.$$

Define $\tilde{\Omega}$ *according to (5) in the main text. Then* $\tilde{\Omega}$ *is the support of* $P_{m,n}(\cdot \mid \boldsymbol{r}, \boldsymbol{c}, \boldsymbol{w})$.

D.3 Alternative treatments of structural zeros

Here we redefine $\boldsymbol{\mathcal{A}}$, U, and V from the main text to account for structural zeros differently. We define $\boldsymbol{\mathcal{A}}$ according to Supplementary Theorem 3 above, which allows trivial cases to be handled by $\tilde{\Omega}$.

Let $\boldsymbol{Y}$ be a random matrix chosen uniformly over the support of P^* and define

$$U(\boldsymbol{x}) = \mathbb{P}(\boldsymbol{Y}^1 = \boldsymbol{x}) = \frac{N_{m,n-1}(\boldsymbol{r} - \boldsymbol{x}, \boldsymbol{c}^{2:n})}{N_{m,n}(\boldsymbol{r}, \boldsymbol{c})}, \qquad V(\boldsymbol{x}) = \mathbb{E}\Big(\prod_{ij} w_{ij}^{Y_{ij}} \Big| \boldsymbol{Y}^1 = \boldsymbol{x}\Big),$$

so that, for any $\boldsymbol{x}$, $P(\boldsymbol{x}) \propto U(\boldsymbol{x})V(\boldsymbol{x})\mathbb{1}\{\boldsymbol{x} \in \Omega\}$. In the main text, these definition were the same except that $\boldsymbol{Y}$ was chosen uniformly over Ω^*. If $\boldsymbol{w}$ forces structural zeros, then the support of P^* may be smaller than Ω^*. We proceed exactly as in the main text to develop approximations of the new U and V.

For the new U, we follow section 4.2 and note that $N_{m,n}(\boldsymbol{r}, \boldsymbol{c})$ needs to be replaced by the size of the support of $P^*_{m,n}(\cdot \mid \boldsymbol{r}, \boldsymbol{c}, \boldsymbol{w})$, say $N_{m,n}(\boldsymbol{r}, \boldsymbol{c}, \boldsymbol{w})$, and, consequently, $\tilde{N}$ needs to be replaced by a combinatorial approximation of the size of the support of P^*. Greenhill & McKay (2009) provide the modified asymptotic enumeration results corresponding to those that led to equation (13) in the main text. Define $a_{ij} = \mathbb{1}\{w_{ij} > 0\}$ for all i, j. Note that $N_{m,n}(\boldsymbol{r}, \boldsymbol{c}, \boldsymbol{w}) = N_{m,n}(\boldsymbol{r}, \boldsymbol{c}, \boldsymbol{a})$. For an approximation of $\tilde{N}_{m,n}(\boldsymbol{r}, \boldsymbol{c}, \boldsymbol{w})$, Greenhill & McKay (2009, Theorem 2.1) suggest

$$\begin{aligned}
\tilde{N}_{m,n}(\boldsymbol{r}, \boldsymbol{c}, \boldsymbol{a}) =& \binom{\sum_{\ell=1}^{m}\sum_{k=1}^{n} a_{\ell k}}{\sum_{k=1}^{n} c_k}^{-1} \prod_{i=1}^{m} \binom{R_i(\boldsymbol{a})}{r_i} \prod_{j=1}^{n} \binom{C_j(\boldsymbol{a})}{c_j} \\
&\times \exp\Big[-\frac{1}{2}\big(1 - \mu_{m,n}(\boldsymbol{r}, \boldsymbol{c})\big)\big(1 - \nu_{m,n}(\boldsymbol{c})\big) - \delta_{m,n}(\boldsymbol{r}, \boldsymbol{c}, \boldsymbol{a})\Big], \\
\delta_{m,n}(\boldsymbol{r}, \boldsymbol{c}, \boldsymbol{a}) =& \eta_{m,n}(\boldsymbol{c}) \sum_{i=1}^{m}\sum_{j=1}^{n} \left((1 - a_{ij}) \Big(r_i - \frac{R_i(\boldsymbol{a})}{mn} \sum_{k=1}^{n} c_k \Big) \Big(c_j - \frac{C_j(\boldsymbol{a})}{mn} \sum_{k=1}^{n} c_k \Big) \right),
\end{aligned}$$

which reduces to the formula in the main text when $\boldsymbol{a} \equiv 1$. The functions μ, ν, η are defined in the main text. This approximation leads to

$$\begin{aligned}
u_i =& \frac{r_i}{R_i(\boldsymbol{a}^{2:n}) - r_i + 1} \exp\Bigg(\eta_{m,n-1}(\boldsymbol{c}^{2:n}) \Big[\big(1 - \nu_{m,n-1}(\boldsymbol{c}^{2:n})\big) \Big(\frac{1}{2} - r_i + \frac{1}{m} \sum_{k=2}^{n} c_k \Big) \\
&+ \sum_{j=2}^{n} (1 - a_{ij}) \Big(c_j - \frac{C_j(\boldsymbol{a})}{m(n-1)} \sum_{k=2}^{n} c_k \Big) \Big] \Bigg),
\end{aligned}$$

where, as before, we set $u_i = 1$ whenever $r_i = 0$ or $r_i = R_i(\boldsymbol{a}^{2:n}) + 1$.

For the new V, we follow section 4.3 in the main text, but define $\boldsymbol{B}$ to be a matrix of independent Bernoulli random variables where B_{ij} is Bernoulli$(a_{ij}/2)$ for $a_{ij} = \mathbb{1}\{w_{ij} > 0\}$.

Following equations (14) and (15) from the main text, the first change comes after the second equality in (15), giving

$$v_i = \frac{w_{i1}\binom{R_i(\boldsymbol{a}^{2:n})}{r_i-1}^{-1} \sum_{\boldsymbol{b}\in\{0,1\}^{n-1}} \mathbb{1}\{\sum_{j=1}^{n-1} b_j = r_i - 1\} \prod_{j=2}^{n} w_{ij}^{b_{j-1}}}{\binom{R_i(\boldsymbol{a}^{2:n})}{r_i}^{-1} \sum_{\boldsymbol{b}\in\{0,1\}^{n-1}} \mathbb{1}\{\sum_{j=1}^{n-1} b_j = r_i\} \prod_{j=2}^{n} w_{ij}^{b_{j-1}}}$$
$$= \frac{w_{i1}(R_i(\boldsymbol{a}^{2:n}) - r_i + 1)G_{n-1}(\boldsymbol{w}_i^{2:n}, r_i - 1)}{r_i G_{n-1}(\boldsymbol{w}_i^{2:n}, r_i)} = \frac{w_{i1}(R_i(\boldsymbol{a}^{2:n}) - r_i + 1)G(i, 2, r_i - 1)}{r_i G(i, 2, r_i)},$$

where we have made use of the fact that terms inside the summations in the first expression are zero whenever $\boldsymbol{b}$ has an entry of one in a place where there is a structural zero. The functions G_n and G are defined above in supplementary section B.3. For zeros in the numerator or denominator of the expression for v_i we set $v_i = 1$ and allow $\mathcal{A}_i$ to deterministically choose the appropriate value of the ith entry. As in the main text, we suggest replacing $\boldsymbol{w}$ with $\bar{\boldsymbol{w}}$ throughout.

E Numerical illustrations

E.1 Pseudorandom number generator

The pseudorandom number generator used by the importance sampling algorithm for the numerical illustrations is the default pseudorandom number generator in Matlab version 7.14, which is the Mersenne twister algorithm `mt19937ar` (c.f. Matsumoto & Nishimura, 1998) described at
`http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html` .

E.2 Canonical weight matrices

The weight matrices $\boldsymbol{w}$ used in Section 6 of the main text are built from a canonical matrix $\boldsymbol{y}$. The $m \times n$ canonical matrix $\boldsymbol{y}$ is constructed as follows:

$$y_{ij} = \frac{R((j-1)m + i)}{2^{31} - 1} \qquad (i = 1, \ldots, m;\ j = 1, \ldots, n),$$

where $R(0) = 1$ and

$$R(k) = 7^5 R(k-1) \bmod (2^{31} - 1) \qquad (k = 1, \ldots, mn).$$

The sequence $R(1), R(2), \ldots$ is a simple, well-known multiplicative congruential pseudorandom number generator, known as MINSTD, for the discrete uniform distribution over $\{1, \ldots, 2^{31} - 2\}$ (Park & Miller, 1988). It was the default pseudorandom number generator in Matlab for many years and is fine for our purpose of creating a matrix $\boldsymbol{y}$ with independent uniform$(0, 1)$ entries whose values are easy to communicate to others.

E.3 The number of $n \times n$ two-regular binary matrices

Anand et al. (1966, Eq. (27)) give a simple recursive formula for the number of $n \times n$ two-regular binary matrices, say H_n. Initialize $H_1 = 0$, $H_2 = 1$, $H_3 = 6$, and then

$$H_k = \frac{1}{2}k(k-1)^2\big((2k-3)H_{k-2} + (k-2)^2 H_{k-3}\big) \qquad (k = 4, 5, \ldots).$$

The exact value of H_{500} can be found in the appendix of this supplement. As noted in Section 6 of the main text, our algorithm provides an extremely accurate approximation.

Chen et al. (2005) used their importance sampling algorithm to approximate H_{100} as $(2.96 \pm 0.03)\times 10^{314}$ based on a sample size of 100. For comparison, using a sample of size 100 from our algorithm gives an approximation of $(2.969 \pm 0.001)\times 10^{314}$, which appears to be almost 1000 times more efficient for the purposes of approximate enumeration. The true value is $2.9692\ldots \times 10^{314}$. The full number can be found in the appendix of this supplement. We should also note that the importance sampling approximations are much more accurate than the combinatorial approximations upon which the importance sampling algorithm is based. For instance, using the approximation $\tilde{N}$ from Section 4.2 of the main text gives 2.957×10^{314}.

E.4 Approximating α-permanents

Here we report comparisons between using our algorithm for approximating α-permanents and using the custom importance sampling algorithm of Kou & McCullagh (2009). We thank Sam Kou for sharing his code with us. The α-permanent of $\boldsymbol{w}$ can be expressed as

$$\mathrm{per}_\alpha(\boldsymbol{w}) = \kappa\, \mathbb{E}(\alpha^{\mathrm{cyc}(\boldsymbol{Z})}),$$

where $\boldsymbol{Z}$ has distribution P^* with the same $\boldsymbol{w}$ and all row and column sums equal to one; see equation (3) in the main text. We approximate it using the consistent, unbiased approximation

$$\hat{\mathrm{per}}_{\alpha,T}(\boldsymbol{w}) = \hat{\kappa}_T \hat{\mu}_T = \frac{1}{T}\sum_{t=1}^{T} f(\boldsymbol{Z}_t) h(\boldsymbol{Z}_t)$$

for $h(\boldsymbol{z}) = \alpha^{\mathrm{cyc}(\boldsymbol{z})}$; see Section 5.2 and equation (17) in the main text.

The Kou & McCullagh algorithm does not attempt to generate $\boldsymbol{Z}$ from a distribution that is close to P^*, like ours does, but rather from a distribution proportional to $h(\boldsymbol{z})P^*(\boldsymbol{z})$. In the case where $\alpha = 1$ so that $h \equiv 1$, the two approaches agree and the empirical results are quite similar. But when $\alpha \neq 1$, their algorithm is generally better, because is it tailored for the choice of α. Nevertheless, our algorithm might be useful in cases where $\mathrm{per}_\alpha(\boldsymbol{w})$ is needed for many α simultaneously, or in cases where α is very close to one.

Supplementary Table 3 reports $\hat{\mathrm{per}}_{\alpha,T}(\boldsymbol{w})$ along with approximate standard errors defined as $\hat{\sigma}_T/\sqrt{T}$, where

$$\hat{\sigma}_T^2 = \frac{1}{T-1}\sum_{t=1}^{T}\big(f(Z_t)h(Z_t) - \hat{\kappa}_T\hat{\mu}_T\big)^2.$$

It also reports an approximate relative standard error defined as

$$\hat{\mathrm{rel}}_T = \frac{\hat{\sigma}_T/\sqrt{T}}{\hat{\kappa}_T\hat{\mu}_T} \times 100\%.$$

We use $T = 1000$ for the examples with $n = 500$ to match Table 1 in the main text. The other $\boldsymbol{w}$ are taken from Kou & McCullagh (2009) and we use $T = 20000$ to facilitate comparison with their results. In some cases the true value of $\mathrm{per}_\alpha(\boldsymbol{w})$ is known and this is shown in the final column of the table; see the supplementary appendix. Except for the $n = 500$ examples and the results from our algorithm, the entries of Supplementary Table 3 come directly from Table 1 in Kou & McCullagh (2009).

Table 3: Approximating α-permanents

parameters			our algorithm		Kou & McCullagh		true value
$\boldsymbol{w}$	n	α	$\hat{\text{per}}_{\alpha,T}(\boldsymbol{w})$	$\hat{\text{rel}}_T\%$	$\hat{\text{per}}_{\alpha,T}(\boldsymbol{w})$	$\hat{\text{rel}}_T\%$	$\text{per}_\alpha(\boldsymbol{w})$
I	500	1	$(1.220 \pm 0.000) \times 10^{1134}$	0.00	$(1.220 \pm 0.000) \times 10^{1134}$	0.00	1.220×10^{1134}
II	500	1	$(1.437 \pm 0.001) \times 10^{1222}$	0.08	$(1.441 \pm 0.001) \times 10^{1222}$	0.08	?
III	500	1	$(3.998 \pm 0.028) \times 10^{983}$	0.69	$(3.975 \pm 0.033) \times 10^{983}$	0.82	?
IV	500	1	$(3.523 \pm 0.056) \times 10^{1133}$	1.60	$(3.546 \pm 0.066) \times 10^{1133}$	1.85	?
I	500	1/2	$(2.963 \pm 0.167) \times 10^{1132}$	5.62	$(3.078 \pm 0.000) \times 10^{1132}$	0.00	3.078×10^{1132}
II	500	1/2	$(3.296 \pm 0.174) \times 10^{1220}$	5.28	$(3.662 \pm 0.012) \times 10^{1220}$	0.32	?
III	500	1/2	$(8.889 \pm 0.459) \times 10^{981}$	5.16	$(1.021 \pm 0.012) \times 10^{982}$	1.21	?
IV	500	1/2	$(9.759 \pm 0.650) \times 10^{1131}$	6.66	$(9.228 \pm 0.228) \times 10^{1131}$	2.47	?
A_1	20	1	$(9.800 \pm 0.008) \times 10^{32}$	0.08	$(9.787 \pm 0.014) \times 10^{32}$	0.14	9.784×10^{32}
A_2	20	1	$(3.513 \pm 0.004) \times 10^{32}$	0.10	$(3.506 \pm 0.007) \times 10^{32}$	0.21	3.514×10^{32}
A_3	15	1/2	$(1.456 \pm 0.009) \times 10^{22}$	0.60	$(1.437 \pm 0.003) \times 10^{22}$	0.22	1.439×10^{22}
A_4	15	1/2	$(7.049 \pm 0.044) \times 10^{21}$	0.63	$(7.043 \pm 0.022) \times 10^{21}$	0.32	7.034×10^{21}
A_5	20	1	$(3.290 \pm 0.003) \times 10^{49}$	0.09	$(3.294 \pm 0.012) \times 10^{49}$	0.38	3.290×10^{49}
A_6	20	1	$(5.928 \pm 0.024) \times 10^{40}$	0.40	$(5.782 \pm 0.103) \times 10^{40}$	1.72	5.946×10^{40}
A_7	15	1/2	$(2.069 \pm 0.013) \times 10^{31}$	0.63	$(2.092 \pm 0.008) \times 10^{31}$	0.37	2.095×10^{31}
A_8	15	1/2	$(1.579 \pm 0.020) \times 10^{25}$	1.27	$(1.549 \pm 0.027) \times 10^{25}$	1.68	1.579×10^{25}
$K(x)_9$	9	1/2	$(4.524 \pm 0.036) \times 10^{0}$	0.79	$(4.504 \pm 0.020) \times 10^{0}$	0.43	4.505×10^{0}
$K(x)_{11}$	11	1/2	$(1.634 \pm 0.014) \times 10^{2}$	0.86	$(1.622 \pm 0.009) \times 10^{2}$	0.56	1.623×10^{2}
$K(x)_{13}$	13	1/2	$(5.815 \pm 0.050) \times 10^{3}$	0.86	$(5.844 \pm 0.026) \times 10^{3}$	0.45	5.816×10^{3}
$K(x)_{15}$	15	1/2	$(2.134 \pm 0.019) \times 10^{5}$	0.89	$(2.117 \pm 0.011) \times 10^{5}$	0.53	2.114×10^{5}
$K(x)_{100}^{Tr}$	100	1/2	$(1.876 \pm 0.118) \times 10^{-16}$	6.28	$(1.928 \pm 0.037) \times 10^{-16}$	1.90	1.911×10^{-16}

E.5 Additional numerical illustrations for the uniform distribution

Our original interest in these problems was motivated by the uniform distribution over Ω^* and we have a variety of simulations investigating this special case. This section is largely reproduced from one of our 2009 unpublished preprints, `arXiv:0906.1004v1`, which focused on comparing different combinatorial approximations and was the basis for our emphasis on the Canfield et al. (2008) approximation in the main text. The simulations from this section were carried out in 2009 on a MacBook laptop with 2 GB of RAM and a 2.16 GHz dual core processor using Matlab. Everything in this section refers to the uniform distribution with $\boldsymbol{w} \equiv 1$.

Supplementary Table 4 details the speed of the algorithm on 1000×1000 binary matrices with all row and column sums identical. These run-times are merely meant to provide a feel for how the algorithm behaves — no attempt was made to control the other processes operating simultaneously on the laptop. Presumably a careful C or assembly language implementation would run much faster. The observed runtime scales closely with the computational complexity of $O(md)$. So, for example, 100×100 r_1-regular matrices can be sampled about 100 times faster than 1000×1000 r_1-regular matrices, and 10×10 matrices can be sampled about 10 000 times faster.

Table 4: Sampling time per 1000×1000 r_1-regular matrix

r_1	2	4	8	16	32	64	128	256	512
time (s)	1.2	1.6	2.4	4.0	6.7	12.4	24.4	39.2	46.6

Supplementary Table 5 reports diagnostics on these examples using $T = 1000$. We

note that the true number of 1000×1000 two-regular matrices is $1.75147\cdots\times10^{5133}$; see Supplementary Section E.3. The approximation from the first row of Supplementary Table 5 is quite accurate.

Table 5: Performance for the uniform distribution over 1000×1000 r_1-regular binary matrices

r_1	$\hat{\Delta}_T$	$\hat{\text{CV}}_T^2$	$\widehat{\kappa}_T$
2	0.049	4.2×10^{-6}	$(1.75148 \pm 0.00011) \times 10^{5133}$
4	0.075	6.4×10^{-6}	$(7.64296 \pm 0.00061) \times 10^{9910}$
8	0.041	2.1×10^{-6}	$(1.01879 \pm 0.00005) \times 10^{18531}$
16	0.008	3.9×10^{-7}	$(2.31580 \pm 0.00005) \times 10^{33629}$
32	0.005	2.3×10^{-7}	$(6.50167 \pm 0.00010) \times 10^{59218}$
64	0.004	2.2×10^{-7}	$(1.22048 \pm 0.00002) \times 10^{100716}$
128	0.004	1.8×10^{-7}	$(9.38861 \pm 0.00013) \times 10^{163302}$
256	0.004	2.2×10^{-7}	$(6.70630 \pm 0.00010) \times 10^{243964}$
512	0.004	2.2×10^{-7}	$(5.02208 \pm 0.00007) \times 10^{297711}$

Bezáková et al. (2006) investigates the performance of the Chen et al. (2005) algorithm on pathological margins with very large r_1 and c_1, but with all other row and column sums exactly 1. They prove that the Chen et al. (2005) proposal distribution is extremely far from uniform for such margins, too far for importance sampling to be practical. It seems likely that our Q^* suffers from the same problem, because of the similarities between the combinatorial approximations in each approach. The empirical performance of the Q^* from the main text is quite bad in these cases; see below. On the other hand, it is straightforward to show that using the combinatorial approximations in Supplementary Section C gives $Q^* = P^*$ for these types of margins.

Following Bezáková et al. (2006), we experiment with the margins $\boldsymbol{r}^{\mathrm{T}} = (240, 1, \ldots, 1)$ and $\boldsymbol{c} = (179, 1, \ldots, 1)$ for a 240×301 matrix. By conditioning on the entry in the first row and the first column and then using symmetry, one can see that

$$N_{m,n}(\boldsymbol{r}, \boldsymbol{c}) = \binom{300}{240}\binom{239}{179}60! + \binom{300}{239}\binom{239}{178}61! = 9.6843\ldots\times10^{205}.$$

Generating a single observation takes about 0.077 s. Using $T = 10^5$ gives $\hat{\Delta}_T = 4.1\times10^{11}$, $\hat{\text{CV}}_T^2 = 1.7\times10^3$, and $\hat{\kappa}_T = (2.2 \pm 0.3)\times10^{205}$, which is quite bad and highly misleading: approximate 95% confidence intervals created by doubling the standard errors would not come close to covering the true value of κ. Alternatively, using the algorithm with u_i from Supplementary Section C gives $\hat{\Delta}_T = \hat{\text{CV}}^2 = 0$ and $\hat{\kappa}_T = \kappa = 9.6843\cdots\times10^{205}$, since $Q^* = P^*$ in this case. In most practical examples, however, the algorithm presented in the main text is superior.

Finally, consider Darwin's finch data (c.f. Chen et al., 2005) which is a 13×17 occurrence matrix with $\boldsymbol{r}^{\mathrm{T}} = (14, 13, 14, 10, 12, 2, 10, 1, 10, 11, 6, 2, 17)$ and $\boldsymbol{c} = (4, 4, 11, 10, 10, 8, 9, 10, 8, 9, 3, 10, 4, 7, 9, 3, 3)$. A single sample takes about 0.001 s. With $T = 10^6$, we find $\hat{\Delta}_T = 2.8\times10^3$ and $\hat{\text{CV}}_T^2 = 0.44$ with $\hat{\kappa}_T = (6.722 \pm 0.004)\times10^{16}$. Chen et al. (2005) report the true value of $\kappa = 67\,149\,106\,137\,567\,626$, and they also report a $\hat{\text{CV}}_T^2$ of "around one" for their algorithm on this problem. Generally speaking, these importance sampling algorithms tend to be less uniform for small irregular problems like this one, than for the larger and/or more regular examples above.

The previous experiments are based primarily on the internal diagnostics of samples from the proposal distribution Q^*. Other than the asymptotic analysis in Blanchet (2009) concerning approximate enumeration using a variation of the algorithm of Chen et al. (2005), there are no external checks on the uniformity of Q^*. Using a complicated, high dimensional proposal distribution without external checks can be dangerous. Indeed, consider the following worst-case scenario. Suppose that $\Omega^* = E \cup E^c$, where E is much smaller than E^c, and suppose that Q^* is uniform over each of E and E^c, but far from uniform over Ω^*, namely,

$$Q^*(\boldsymbol{z}) = \frac{1-\epsilon}{|E|}\mathbb{1}\{\boldsymbol{z} \in E\} + \frac{\epsilon}{|E^c|}\mathbb{1}\{\boldsymbol{z} \in E^c\} \qquad (|E|/|E^c| \ll \epsilon \ll 1).$$

If ϵ is extremely tiny, say $\epsilon = 10^{-100}$, then Monte Carlo samples from Q^* will, practically speaking, always lie in E, which itself is a tiny fraction of Ω^*. Furthermore, all internal diagnostics will report that Q^* is exactly uniform, since it is uniform over E. But, of course, statistical inferences based on samples from Q^* will tend to be completely wrong. This section describes two types of experiments designed to provide external checks on the uniformity of Q^*.

For the first set of experiments we generate a binary matrix $\boldsymbol{Z}$ from the uniform distribution over all binary matrices with row sums $\boldsymbol{r}$. This is easy to do by independently and uniformly choosing each row of $\boldsymbol{Z}$ from one of the $\binom{n}{r_i}$ possible configurations. Since the conditional distribution of $\boldsymbol{Z}$ given its columns sums $\boldsymbol{C}$ is uniform over $\Omega^*_{m,n}(\boldsymbol{r}, \boldsymbol{C})$, we can view $\boldsymbol{Z}$ as a single observation from the uniform distribution over $\Omega^*_{m,n}(\boldsymbol{r}, \boldsymbol{C})$. Of course, there is no practical way to uniformly and independently generate another such $\boldsymbol{Z}$ with the same $\boldsymbol{C}$. Notice that the importance weight $f(\boldsymbol{Z})$ gives external information about the uniformity of Q^* for these margins, since it gives the value of $1/Q^*$ at a uniformly chosen location in $\Omega^*_{m,n}(\boldsymbol{r}, \boldsymbol{C})$. Indeed, in the pathological thought experiment described above, $\boldsymbol{Z}$ would almost certainly be in E^c and $f(\boldsymbol{Z})$ would be substantially larger than any of the importance weights. Alternatively, if Q^* is nearly uniform, then $f(\boldsymbol{Z})$ should be indistinguishable from the other importance weights. In summary, we can compare Q^* to P^* by comparing the importance weights to $f(\boldsymbol{Z})$. This observation can also be used to give valid Monte Carlo p-values with importance sampling, even if the importance sampling distribution is far from the target distribution (Harrison, 2012).

Each experiment of this type proceeds identically. We fix m, n, and $\boldsymbol{r}$. Then we generate L iid observations, say $\boldsymbol{Z}_0^{(1)}, \ldots, \boldsymbol{Z}_0^{(L)}$, from the uniform distribution over all $m \times n$ binary matrices with row sums $\boldsymbol{r}$. The column sums of these matrices are $\boldsymbol{C}^{(1)}, \ldots, \boldsymbol{C}^{(L)}$. Then, for each $\ell = 1, \ldots, L$, we generate T iid observations, say $\boldsymbol{Z}_1^{(\ell)}, \ldots, \boldsymbol{Z}_T^{(\ell)}$, from the proposal distribution Q^* over $\Omega^*_{m,n}(\boldsymbol{r}, \boldsymbol{C}^{(\ell)})$. We compute the ratio of maximum to minimum importance weights *including the original observation* for each ℓ, namely,

$$\hat{\Delta}_T^{(\ell)} = \frac{\max_{t=0,\ldots,T} f(\boldsymbol{Z}_t^{(\ell)})}{\min_{t=0,\ldots,T} f(\boldsymbol{Z}_t^{(\ell)})} - 1,$$

and we report the final summary $\hat{\Delta}_T^{\max} = \max_{\ell=1,\ldots,L} \hat{\Delta}_T^{(\ell)}$. If $\hat{\Delta}_T^{\max}$ is close to zero, then this provides evidence that Q^* is approximately uniform over a large part of each $\Omega^*_{m,n}(\boldsymbol{r}, \boldsymbol{C}^{(\ell)})$.

We begin with 1000×1000 matrices with regular row sums $r_1 = \cdots = r_m$, but the column sums will not be regular, since they are generated randomly. We use $L = 10$ and

$T = 10$ for the cases $r_1 = 2, 8, 32$, finding $\hat{\Delta}_T^{\max} = 0.0002, 0.0023, 0.0051$, respectively. For another example, take the row sums for the irregular 50×100 case that was used for Table 2 in the main text and take $k = 1$, i.e., $\boldsymbol{r} = \tilde{\boldsymbol{r}}$. We use $L = 100$ and $T = 1000$ and find that $\hat{\Delta}_T^{\max} = 1.264$. These preliminary experiments are encouraging, and suggest that Q^* is indeed a good approximation of uniform P^* in many cases.

For the second type of experiment, we try to design an extreme $\boldsymbol{z}' \in \Omega^*$ and compare the importance weights to $f(\boldsymbol{z}')$. Again, if Q^* is approximately uniform over all of Ω^* then $f(\boldsymbol{z}')$ should be indistinguishable from the other importance weights. For these experiments we report

$$\hat{\Delta}'_T = \frac{\max\{f(\boldsymbol{z}'), f(\boldsymbol{Z}_1), \ldots, f(\boldsymbol{Z}_T)\}}{\min\{f(\boldsymbol{z}'), f(\boldsymbol{Z}_1), \ldots, f(\boldsymbol{Z}_T)\}} - 1,$$

which should be close to zero if the region in Ω^* where Q^* is approximately uniform includes $\boldsymbol{z}'$.

Consider the regular case where $m = n = 1000$ and $r_1 = r_i = c_j$ for all i, j. Suppose that r_1 evenly divides 1000 and let $\boldsymbol{z}'$ be comprised only of disjoint $r_1 \times r_1$ blocks of ones. In particular, take $\boldsymbol{z}'_{ij} = 1$ for $(k-1)r_1+1 \leq i, j \leq kr_1$ and for $k = 1, \ldots, 1000/r_1$. For the cases $r_1 = 2, 4, 8$ we compute $f(\boldsymbol{z}')$ and compare it to the data that generated the corresponding parts of table 5, obtaining $\hat{\Delta}'_T = 0.741, 26.24, 6.25 \times 10^4$, respectively. Clearly, Q^* is not a uniformly accurate approximation of P^* over all of Ω^* and is unlikely to be useful as a proposal distribution for rejection sampling to get exact samples from P^*. Nevertheless, Q^* seems to be extremely well-suited as a proposal distribution for importance sampling. For another example, consider the irregular 50×100 case that was used for Table 2 in the main text and take $k = 1$, i.e., $\boldsymbol{r} = \tilde{\boldsymbol{r}}$ and $\boldsymbol{c} = \tilde{\boldsymbol{c}}$. We construct a pathological $\boldsymbol{z}'$ as follows. Place c_1 ones in the last c_1 rows, corresponding to the smallest row sums, of the first column. Place c_2 ones in the last available c_2 rows of the second column, where a row is available if placing a one in that row will not exceed the row sum for that row. Continue in this manner until all the columns are assigned or until a column cannot be assigned successfully. In general, this procedure is not guaranteed to terminate successfully, but it does for this choice of margins. The resulting $\boldsymbol{z}'$ is unusual because rows and columns with large sums tend to have zeros at the intersecting entry. Using the data from the corresponding part of Table 2 in the main text gives $\hat{\Delta}'_T = 14.37$.

Supplementary Appendix

The number of 100×100 two-regular binary matrices

2969298425 4879211020 5463258948 9046531125 6932010720 0899043082 6661472985 5602957737
5386603250 7914169840 3947972542 0803105057 9494091210 8196163985 3132939771 8223074880
1582489734 4113002630 0345104451 5505567811 8301236764 6670284335 5753266570 2919415207
2361422613 1731302283 4023510256 2089359423 4174989926 4000000000 0000000000 00000

The number of 500×500 two-regular binary matrices

2276586004 3872645654 7163822917 6140246378 6529219189 6007058852 1885701633 9308224336
7024859918 5873168947 8428993358 7710991052 6831024823 1020957186 1359882527 3634597638
7751901014 9459428637 5300752209 6236400145 2272455600 2450498447 6886449802 2657577100
8803085437 1426603063 9060350752 5676829379 2441654640 4384402364 9178512515 5701834312
5382285704 7911170936 9213162976 1124369611 0263144354 2492660647 6317501009 4702298551
3783877264 5366936440 0850289755 0247749665 4582496735 4778933695 9359401807 4728987947
4052084791 8351006525 6516882276 6819426986 4276522770 8754690714 8153703130 7689579335
5313886817 9879619523 6757312609 9563935644 5860973860 5720751902 8525628015 1655464790
3607836217 2202522127 9381851238 5339132917 8663772909 4697618230 9562268584 1389355037
4200343275 4426328049 4429348983 4734923700 0635594018 1200043308 9996436581 2082429967
1420144526 3238392163 0625410465 1147246306 0267066287 2838455102 1984436331 4795820153
4878729606 4682614593 4828351763 2549610945 2823414530 6966187549 3636469942 1582542169
0511243887 9654470644 8952801709 4100687806 1803581920 0502635810 6084543151 8196763100
9226192052 8186323173 8128828855 7307283447 5486503911 0996089630 7969624574 8668199425
1430690842 9240854111 3288457886 5068062328 1130147009 2410850737 0194640624 5215023611
0105313331 5631006370 7547904555 8541951209 3762970404 4299114208 6898539174 1261578007
5271576323 7806458898 5197173413 2333790169 8450503603 6175432120 4646913329 9283772618
0789892314 7885014128 9831206980 1470933069 2885920165 3886059912 3547627990 2473766270
0084914243 1261925800 3966112818 5515090740 2869173796 5265773700 6653705150 0776999823
6682749949 6649629337 6729065663 7740220752 0069908832 1026134189 8109544591 4141299020
9944691129 8101632276 5735759559 3131678694 4342947732 7389063830 1146871076 6098180223
5086650691 0193318778 3650834389 5788540935 3233656140 3425148468 8948999361 5539721393
2767810044 6245991329 5809908199 9005968612 6446584189 0334076925 7082772956 3377889631
0446650398 8183375905 5124117054 7434261832 8900372657 5745038153 2952534928 4112016395
9467531245 7165626500 2517876951 1088955612 4288697963 9375087520 6487400471 4382991165
8206541306 8546637026 9648941941 8803223917 8589969888 6361729999 1147924387 2385375087
0828596942 2197021633 2700563010 0820849326 1167561772 1388697124 8640000000 0000000000
0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000000
0000000000 0000000000 0000000

Exactly computing the α-permanent of a constant matrix

If $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n)$ is a permutation chosen uniformly at random and C is the number of disjoint cycles in $\boldsymbol{\pi}$, then C has the same distribution as $B_1 + \cdots + B_n$, where each B_i is independent Bernoulli$(1/i)$ (Durrett, 2010, Lemma 2.2.5). If $\boldsymbol{w}$ is an $n \times n$ constant matrix with common entry b, then

$$\mathrm{per}_\alpha(\boldsymbol{w}) = n! b^n \, \mathbb{E}(\alpha^C) = n! b^n \prod_{i=1}^{n} \mathbb{E}(\alpha^{B_i}) = n! b^n \prod_{i=1}^{n} \Big(\frac{\alpha}{i} + \big(1 - \frac{1}{i}\big) \Big) = n! b^n \prod_{i=1}^{n} \frac{i + \alpha - 1}{i}.$$

We used this formula with $n = 500$ and $b = 1$ to get the true value of $\mathrm{per}_\alpha(\boldsymbol{w})$ for $\boldsymbol{w}$ in class I in Supplementary Table 3.

Matlab implementation of the algorithm

This is a place-holder for cleaner, shorter code that will be inserted prior to publication. Software will also be available on the author's website.

```
function [logQ,logP,alist] = BernoulliMarginsRnd(SampN,rN,cN,wN,pflag,wflag,cflag,bIN)
%function [logQ,logP,alist] = BernoulliMarginsRnd(N,r,c,w,pflag,wflag,cflag,Binput)
%
% Approximate sampling from independent Bernoulli random variables B(i,j)
% arranged as an m x n matrix B given the m-vector of row sums r and the
```

```
% n-vector of column sums c, i.e., given that sum(B,2)=r and sum(B,1)=c.
%
% An error is generated if no binary matrix agrees with r and c.
%
% B(i,j) is Bernoulli(p(i,j)) where p(i,j)=w(i,j)/(1+w(i,j)), i.e.,
% w(i,j)=p(i,j)/(1-p(i,j)).  [The case p(i,j)=1 must be handled by the user
% in a preprocessing step, by converting to p(i,j)=0 and decrementing the
% row and column sums appropriately.]
%
% Use w=[] for w identically 1, i.e., approximate uniform sampling over
% binary matrices with margins r and c.
%
% N is the sample size.  Because of pre-processing, it is more efficient
% per matrix to use larger sample sizes.
%
% alist stores the locations of the ones in the samples.
% If d = sum(r) = sum(c), then alist is 2 x d x N.
%
% The 1-entries of the kth matrix are stored as alist(:,:,k).  The
% (row,column) indices are (alist(1,t,k),alist(2,t,k)) for t=1:d.
%
% If B is the kth matrix, then B can be created from alist via:
%
% B = false(m,n); for t = 1:size(alist,2), B(alist(1,t,k),alist(2,t,k)) = true; end
%
% logQ(k)=log(probability that algorithm generates B)
% logP(k)=log(prod(w(B)))
%
% If the algorithm is used for importance sampling, then the kth
% unnormalized importance weight is exp(logP(k)-logQ(k)).
%
% NOTE for w(i,j)=0:
%
% If the entries of w are not strictly positive, then the algorithm can
% sometimes generate matrices with logP(k)=-inf.  In these cases, some of
% the entries of alist(:,:,k) may be zero and logQ(k) corresponds to the
% probability of generating that particular alist(:,:,k).
%
% OPTIONS:
%
% pflag: 'canfield' or '' (default, works best in most cases)
%        'greenhill' (perhaps useful for sparse and highly irregular margins)
% pflag controls which combinatorial approximations are used
%
% wflag: 'sinkhorn' or '' (default)
% wflag controls the initial balancing of w; it is passed to canonical.m
%
% cflag: 'descend' or '' (default)
%        'none' (sample columns in original order)
% cflag controls the order in which the columns are sampled
%
% Binput is a m x n binary matrix.  If it is provided, then the algorithm
% computes the probability of generating this matrix.

if nargin < 8 || isempty(bIN)
    doIN = false;
else
    doIN = true;
end
if nargin < 7 || isempty(cflag)
    cflag = 'descend';
end
if nargin < 6 || isempty(wflag)
    wflag = 'sinkhorn';
end
if nargin < 5 || isempty(pflag)
    pflag = 'canfield';
end
if nargin < 4
    wN = [];
end

doW = true;
if isempty(wN), doW = false; end

doA = true;
if nargout < 2, doA = false; end

if ~isscalar(SampN) || SampN < 1 || SampN ~= round(SampN), error('SampN must be a positive integer'), end

ptype = 0;
switch lower(pflag)
    case 'canfield'
        ptype = 1;
    case 'greenhill'
        ptype = 2;
    otherwise
        error('unknown pflag')
end
```

```
%-----------------------------------------------------%
%--------------- START: PREPROCESSING ----------------%
%-----------------------------------------------------%

% sizing
mT = numel(rN);
nT = numel(cN);

% sort the marginals (descending)
rT = rN(:);
[rsort,rndxT] = sort(rT,'descend');

if doW
    % balance the weights
    [~,~,wopt] = canonical(wN,wflag);
    % reorder the columns
    switch lower(cflag)
        case 'none'
            cndx = 1:nT;
        case 'descend'
            [~,cndx] = sortrows(-[cN(:) var(wopt,0,1).']);
        otherwise
            error('unknown cflag')
    end
    csort = cN(cndx);
    wopt = wopt(:,cndx);
    % precompute log weights
    logw = log(wN);

    % -----------------------------------------------------
    % precompute G

    logwopt = log(wopt);

    rmax = max(rT);
    G = -inf(rmax+1,mT,nT-1);
    G(1,:,:) = 0;
    G(2,:,nT-1) = logwopt(:,nT);

    for i = 1:mT
        ri = rT(i);
        for j = nT-1:-1:2
            wij = logwopt(i,j);
            for k = 2:ri+1
                b = G(k-1,i,j)+wij;
                a = G(k,i,j);
                if a > -inf || b > -inf
                    if a > b
                        G(k,i,j-1) = a + log(1+exp(b-a));
                    else
                        G(k,i,j-1) = b + log(1+exp(a-b));
                    end
                end
            end
        end

        for j = 1:nT-1
            for k = 1:rmax
                Gknum = G(k,i,j);
                Gkden = G(k+1,i,j);
                if isinf(Gkden)
                    G(k,i,j) = -1;
                else
                    G(k,i,j) = wopt(i,j)*exp(Gknum-Gkden)*((nT-j-k+1)/k);
                end
            end
            if isinf(Gkden)
                G(rmax+1,i,j) = -1;
            end
        end
    end
    % -----------------------------------------------------
else
    switch lower(cflag)
        case 'none'
            cndx = 1:numel(cN);
        case 'descend'
            [csort,cndx] = sort(cN(:),'descend');
        otherwise
            error('unknown cflag')
    end
end

% generate the inverse index for the row orders to facilitate fast
% sorting during the updating
irndxT = (1:mT).'; irndxT(rndxT) = irndxT;

% basic input checking
if rsort(1) > nT || rsort(mT) < 0 || csort(1) > mT || csort(nT) < 0 || any(rsort ~= round(rsort)) || any(csort ~= round(csort))
   error('marginal entries invalid')
```

```
end

% compute the conjugate of c
cconjT = conjugate_local(csort,mT);

% get the running total of number of ones to assign
countT = sum(rsort);

% get the running total of sum of c squared
ccount2T = sum(csort.^2);
% get the running total of (2 times the) column marginals choose 2
ccount2cT = sum(csort.*(csort-1));
% get the running total of (6 times the) column marginals choose 3
ccount3cT = sum(csort.*(csort-1).*(csort-2));

% get the running total of sum of r squared
rcount2T = sum(rsort.^2);
% get the running total of (2 times the) row marginals choose 2
rcount2cT = sum(rsort.*(rsort-1));
% get the running total of (6 times the) row marginals choose 3
rcount3cT = sum(rsort.*(rsort-1).*(rsort-2));

% check for compatible marginals
if countT ~= sum(csort) || any(cumsum(rsort) > cumsum(cconjT)), error('marginal sums invalid'), end

% initialize the memory
logQ = zeros(SampN,1);
logP = zeros(SampN,1);
if doA, AN = SampN; else AN = 1; end
alist = zeros(2,countT,AN);
% initialize the memory
M = csort(1)+3; % index 1 corresponds to -1; index 2 corresponds to 0, index 3 corresponds to 1, ..., index M corresponds to c(1)+1
S = zeros(M,nT);
SS = zeros(M,1);

eps0 = eps(0); % used to prevent divide by zero

%-------------------------------------------------------%
%--------------- END: PREPROCESSING --------------------%
%-------------------------------------------------------%

% loop over the number of samples
for SampLoop = 1:SampN

    %--------------- INITIALIZATION ------------------------%
    if doA, ALoop = SampLoop; else ALoop = 1; end

    % copy in initialization
    r = rT;
    rndx = rndxT;
    irndx = irndxT;

    cconj = cconjT;
    count = countT;
    ccount2 = ccount2T;
    ccount2c = ccount2cT;
    ccount3c = ccount3cT;
    rcount2 = rcount2T;
    rcount2c = rcount2cT;
    rcount3c = rcount3cT;
    m = mT;
    n = nT;

    % initialize
    place = 0; % most recent assigned column in alist
    logq = 0; % running log probability
    logp = 0;

    %-------------------------------------------------------%
    %--------------- START: COLUMN-WISE SAMPLING ----------%
    %-------------------------------------------------------%

    %-------- loop over columns ------------%
    for c1 = 1:nT

        %-------------------------------------------------------------%
        %------------- START: SAMPLE THE NEXT "COLUMN" -------------------%
        %-------------------------------------------------------------%

        % remember the starting point for this columns
        placestart = place + 1;

        %---------------------------------
        % sample a col
        %---------------------------------

        label = cndx(c1); % current column label

        colval = csort(c1); % current column value
```

```
if colval == 0 || count == 0, break, end

% update the conjugate
for i = 1:colval
    cconj(i) = cconj(i)-1;
end
% update the number of columns remaining
n = n - 1;

%------------ DP initialization -----------

smin = colval;
smax = colval;
cumsums = count;
% update the count
count = count - colval;
% update running total of sum of c squared
ccount2 = ccount2 - colval^2;
% update the remaining (two times the) sum of column sums choose 2
ccount2c = ccount2c - colval*(colval-1);
% update the remaining (six times the) sum of column sums choose 3
ccount3c = ccount3c - colval*(colval-1)*(colval-2);

cumconj = count;

SS(colval+3) = 0;
SS(colval+2) = 1;
SS(colval+1) = 0;

% get the constants for computing the probabilities
% it is faster to compute them all, than to check pflag
d = ccount2c/count^2;
if (count == 0) || (m*n == count)
    weightA = 0;
else
    weightA = m*n/(count*(m*n-count));
    weightA = weightA*(1-weightA*(ccount2-count^2/n))/2;
end

d2 = ccount2c/(2*count^2+eps0) + ccount2c/(2*count^3+eps0) + ccount2c^2/(4*count^4+eps0);
d3 = -ccount3c/(3*count^3+eps0) + ccount2c^2/(2*count^4+eps0);
d22 = ccount2c/(4*count^4+eps0) + ccount3c/(2*count^4+eps0) - ccount2c^2/(2*count^5+eps0);

%----------- dynamic programming ----------
SSS = 0;
% loop over (remaining and sorted descending) rows in reverse
for i = m:-1:1

    % get the value of this row and use it to compute the
    % probability of a 1 for this row/column pair
    rlabel = rndx(i);
    val = r(rlabel);
    if ptype == 1
        % canfield
        p = val*exp(weightA*(1-2*(val-count/m)));
        p = p./(n+1-val+p);
        q = 1-p;
    elseif ptype == 2
        % greenhill
        q = 1/(1+val*exp((2*d2+3*d3*(val-2)+4*d22*(rcount2c-val+1))*(val-1)));
        p = 1-q;
    else
        % never get here
        p = 0; q = 0; % helps compiler
    end

    % incorporate weights
    if doW && n > 0 && val > 0
        Gk = G(val,rlabel,c1);
        if Gk < 0
            q = 0;
        else
            p = p*Gk;
        end
    end

    % update the feasibility constraints
    cumsums = cumsums - val;
    cumconj = cumconj - cconj(i);

    sminold = smin;
    smaxold = smax;

    % incorporate the feasibility constraints into bounds on the
    % running column sum
    smin = max(0,max(cumsums-cumconj,sminold-1));
    smax = min(smaxold,i-1);

    % DP iteration
    SSS = 0;
```

```
    SS(smin+1) = 0;  % no need to set S(1:smin) = 0, since it is not accessed
    for j = smin+2:smax+2
        a = SS(j)*q;
        b = SS(j+1)*p;
        apb = a + b;
        SSS = SSS + apb;
        SS(j) = apb;
        S(j,i) = b/(apb+eps0);
    end
    SS(smax+3) = 0;  % no need to set S(smax+4:end) = 0, since it is not accessed

    % check for impossible
    if SSS <= 0, break, end

    % normalize to prevent overflow/underflow
    for j = smin+2:smax+2
        SS(j) = SS(j) / SSS;
    end

end

% check for impossible
if SSS <= 0, logp = -inf; break, end

%---------- sampling ----------
j = 2; % running total (offset to match indexing offset)
jmax = colval + 2;
if j < jmax % skip assigning anything when colval == 0
    if doIN
        for i = 1:m
            % get the transition probability of generating a one
            p = S(j,i);
            % get the current row
            rlabel = rndx(i);
            if bIN(rlabel,label)

                % if we have a generated a one, then decrement the current
                % row total
                val = r(rlabel);
                r(rlabel) = val-1;

                rcount2 = rcount2 - 2*val + 1;
                rcount2c = rcount2c - 2*val + 2;
                rcount3c = rcount3c - 3*(val-1)*(val-2);

                % record the entry and update the log probability
                place = place + 1;
                logq = logq + log(p);
                if doW, logp = logp + logw(rlabel,label); end
                alist(1,place,ALoop) = rlabel;
                alist(2,place,ALoop) = label;
                j = j + 1;
                % the next test is not necessary, but seems more efficient
                % since all the remaining p's must be 0
                if j == jmax, break, end
            else
                logq = logq + log(1-p);
            end
        end
    else
        for i = 1:m
            % get the transition probability of generating a one
            p = S(j,i);
            if rand <= p

                % if we have a generated a one, then decrement the current row total
                rlabel = rndx(i);
                val = r(rlabel);
                r(rlabel) = val-1;

                rcount2 = rcount2 - 2*val + 1;
                rcount2c = rcount2c - 2*val + 2;
                rcount3c = rcount3c - 3*(val-1)*(val-2);

                % record the entry and update the log probability
                place = place + 1;
                logq = logq + log(p);
                if doW, logp = logp + logw(rlabel,label); end
                alist(1,place,ALoop) = rlabel;
                alist(2,place,ALoop) = label;
                j = j + 1;
                % the next test is not necessary, but seems more efficient
                % since all the remaining p's must be 0
                if j == jmax, break, end
            else
                logq = logq + log(1-p);
            end
        end
    end
```

```
        end

        %-------------------------------------------------------------------%
        %------------ END: SAMPLE THE NEXT "COLUMN" ----------------------%
        %-------------------------------------------------------------------%

        if count == 0, break, end

        %-----------------------------------------------
        % everything is updated except the sorting
        %-----------------------------------------------

        %-------------------------------------------------------------------%
        %------------ START: RESORT THE NEW ROW SUMS ----------------------%
        %-------------------------------------------------------------------%

        % re-sort the assigned rows

        % this code block takes each row that was assigned to the list
        % and either leaves it in place or swaps it with the last row
        % that matches its value; this leaves the rows sorted (descending)
        % since each row was decremented by only 1

        % looping in reverse ensures that least rows are swapped first
        for j = place:-1:placestart
            % get the row label and its new value (old value -1)
            k = alist(1,j,ALoop);
            val = r(k);
            % find its entry in the sorting index
            irndxk = irndx(k);
            % look to see if the list is still sorted
            irndxk1 = irndxk + 1;
            if irndxk1 > m || r(rndx(irndxk1)) <= val
                % no need to re-sort
                continue;
            end
            % find the first place where k can be inserted
            irndxk1 = irndxk1 + 1;
            while irndxk1 <= m && r(rndx(irndxk1)) > val
                irndxk1 = irndxk1 + 1;
            end
            irndxk1 = irndxk1 - 1;
            % now swap irndxk and irndxk1
            rndxk1 = rndx(irndxk1);
            rndx(irndxk) = rndxk1;
            rndx(irndxk1) = k;
            irndx(k) = irndxk1;
            irndx(rndxk1) = irndxk;
        end

        %-------------------------------------------------------------------%
        %------------ END: RESORT THE NEW ROW SUMS ----------------------%
        %-------------------------------------------------------------------%

        % r(rndx(rndx1:rndxm)) is sorted descending and has exactly those
        % unassigned rows
        % rndx(rndx1:rndxm) still gives the labels of those rows
        % rndx(irndx(k)) = k
        %
        % c(c1+1:cn) is sorted descending and has exactly those unassigned columns
        % cndx(c1+1:cn) still gives the labels of those columns
        %
        % m, n, count, ccount2, ccount2c are valid for the remaining rows, cols

    end

    logQ(SampLoop) = logq;
    logP(SampLoop) = logp;

end

%-------------------------------------------------------------------------%
%-------------------------------------------------------------------------%
%-------------------------------------------------------------------------%
%------------------ END OF MAIN FUNCTION ---------------------------------%
%-------------------------------------------------------------------------%
%-------------------------------------------------------------------------%
%-------------------------------------------------------------------------%

% helper function (just to keep everything together... not for efficiency,
% since it is only called once)

function cc = conjugate_local(c,n)
% function cc = conjugate(c,n)
%
% let c(:) be nonnegative integers
% cc(k) = sum(c >== k)  for k = 1:n

cc = zeros(n,1);
```

```
%c = min(c,n);

for j = 1:numel(c)
    k = c(j);
    if k >= n
        cc(n) = cc(n) + 1;
    elseif k >= 1
        cc(k) = cc(k) + 1;
    end
end

s = cc(n);
for j = n-1:-1:1
    s = s + cc(j);
    cc(j) = s;
end

%-----------------------------------

function [a,b,abw,k] = canonical(w,flag,tol,maxiter,r,c)

[m,n] = size(w);

if nargin <6 || isempty(c)
c = ones(1,n);
elseif size(c,1) ~= 1
c = c(:).';
end
if nargin <5 || isempty(r)
r = ones(m,1);
elseif size(r,2) ~= 1
r = r(:);
end
if nargin <4 || isempty(maxiter)
    maxiter = 10^5;
end
if nargin <3 || isempty(tol)
    tol = 1e-8;
end
if nargin <2 || isempty(flag)
    flag = 'sinkhorn';
end

switch lower(flag)

    case 'sinkhorn'

M = sum(w>0,1); N = sum(w>0,2);
a = N./sum(w,2); a = a/mean(a);
b = M./sum(bsxfun(@times,a,w),1);

if tol >= 0, a0 = a; b0 = b; end

        k = 0;
        tolcheck = inf;
        while k < maxiter && tolcheck > tol
            k = k + 1;

a = N./sum(bsxfun(@times,b,w),2); a = a/mean(a);
b = M./sum(bsxfun(@times,a,w),1);

if tol >= 0
                tolcheck = sum(abs(a-a0))+sum(abs(b-b0));
a0 = a; b0 = b;
end
        end

    case 'sinkhorn-col'

w = fliplr(w);

M = sum(w>0,1); N = cumsum(w>0,2);
aa = N./cumsum(w,2);
b = M./sum(w.*aa,1); b = b / mean(b);
a = aa(:,n);

if tol >= 0, a0 = a; b0 = b; end

        k = 0;
        tolcheck = inf;
        while k < maxiter && tolcheck > tol
            k = k + 1;

aa = N./cumsum(bsxfun(@times,b,w),2);
b = M./sum(w.*aa,1); b / mean(b);
a = aa(:,n);

if tol >= 0
                tolcheck = sum(abs(a-a0))+sum(abs(b-b0));
a0 = a; b0 = b;
```

```
end
end

w = fliplr(w);
b = fliplr(b);

case 'log'

w0 = w > 0;
M = sum(w0,1); N = sum(w0,2);
logw = log(w+~w0);
a = exp(-sum(logw,2)./N);
b = exp(-sum(logw,1)./M);

case 'entropy'

w1 = (w > 0)./max(w,eps(0));
a = sqrt(sum(w1,2)./sum(w,2)); a = a/mean(a);
b = sqrt(sum(bsxfun(@rdivide,w1,a),1)./sum(bsxfun(@times,a,w),1));

if tol >= 0, a0 = a; b0 = b; end

        k = 0;
        tolcheck = inf;
        while k < maxiter && tolcheck > tol
            k = k + 1;

a = sqrt(sum(bsxfun(@rdivide,w1,b),2)./sum(bsxfun(@times,b,w),2)); a = a/mean(a);
b = sqrt(sum(bsxfun(@rdivide,w1,a),1)./sum(bsxfun(@times,a,w),1));

if tol >= 0
                tolcheck = sum(abs(a-a0))+sum(abs(b-b0));
a0 = a; b0 = b;
end
        end

case 'l2'

w2 = w.^2;

a = sum(w,2)./sum(w2,2); a = a/mean(a);
b = sum(bsxfun(@times,a,w),1)./sum(bsxfun(@times,a.^2,w2),1);

if tol >= 0, a0 = a; b0 = b; end

        k = 0;
        tolcheck = inf;
        while k < maxiter && tolcheck > tol
            k = k + 1;

a = sum(bsxfun(@times,b,w),2)./sum(bsxfun(@times,b.^2,w2),2); a = a/mean(a);
b = sum(bsxfun(@times,a,w),1)./sum(bsxfun(@times,a.^2,w2),1);

if tol >= 0
                tolcheck = sum(abs(a-a0))+sum(abs(b-b0));
a0 = a; b0 = b;
end
        end

    case 'l2p'

        w2 = w.^2;

        c = (1+w).^3;
a = sum(w./c,2)./sum(w2./c,2); a = a/mean(a);
        c = (1+bsxfun(@times,a,w)).^3;
b = sum(bsxfun(@times,a,w)./c,1)./sum(bsxfun(@times,a.^2,w2)./c,1);

if tol >= 0, a0 = a; b0 = b; end

        k = 0;
        tolcheck = inf;
        while k < maxiter && tolcheck > tol
            k = k + 1;

            c = (1+a*b.*w).^3;
a = sum(bsxfun(@times,b,w)./c,2)./sum(bsxfun(@times,b.^2,w2)./c,2); a = a/mean(a);
            c = (1+a*b.*w).^3;
b = sum(bsxfun(@times,a,w)./c,1)./sum(bsxfun(@times,a.^2,w2)./c,1);

if tol >= 0
                tolcheck = sum(abs(a-a0))+sum(abs(b-b0));
a0 = a; b0 = b;
end
        end

    case 'ratio'

        wz = w > 0;
        w(~wz) = eps(0);
```

```
        a = sqrt(sum(wz./w,2)./sum(w,2)); a = a/mean(a);
b = sqrt(sum(wz./(bsxfun(@times,a,w)),1)./sum(bsxfun(@times,a,w),1));

if tol >= 0, a0 = a; b0 = b; end

        k = 0;
        tolcheck = inf;
        while k < maxiter && tolcheck > tol
            k = k + 1;

a = sqrt(sum(wz./(bsxfun(@times,b,w)),2)./sum(bsxfun(@times,b,w),2)); a = a/mean(a);
            b = sqrt(sum(wz./(bsxfun(@times,a,w)),1)./sum(bsxfun(@times,a,w),1));

if tol >= 0
                tolcheck = sum(abs(a-a0))+sum(abs(b-b0));
a0 = a; b0 = b;
end
        end

case 'barvinok'

s = log(r/n);
t = log(c/m);

M = w.*(exp(s)*exp(t));
M = M ./ (1+M);

sMr = sum(M,2)-r;
sMc = sum(M,1)-c;

tolcheck = sum(abs(sMr))+sum(abs(sMc));

alpha = .01;

while tolcheck > tol

s = s - alpha*sMr;
t = t - alpha*sMc;

M = w.*(exp(s)*exp(t));
M = M ./ (1+M);

sMr = sum(M,2)-r;
sMc = sum(M,1)-c;

tolcheck = sum(abs(sMr))+sum(abs(sMc));
end

a = exp(s);
b = exp(t);

    otherwise

        error('unknown flag')
end

if nargout > 2, abw = a*b.*w; end
```

References

Admiraal, R., & Handcock, M. S. (2008). networksis: a package to simulate bipartite graphs with fixed marginals through sequential importance sampling. *J. Statist. Software*, *24*(8), 1–21.

Anand, H., Dumir, V. C., & Gupta, H. (1966). A combinatorial distribution problem. *Duke Math. J.*, *33*(4), 757–769.

Ando, T. (1989). Majorization, doubly stochastic matrices, and comparison of eigenvalues. *Linear Algebra Appl.*, *118*, 163–248.

Bapat, R. B., & Beg, M. I. (1989). Order statistics for nonidentically distributed variables and permanents. *Sankhyā Ser. A*, *51*, 79–93.

Barvinok, A. (2010a). Matrices with prescribed row and column sums. *Linear Algebra Appl.*, *436*, 820–834.

Barvinok, A. (2010b). On the number of matrices and a random matrix with prescribed row and column sums and 0-1 entries. *Adv. Math.*, *224*(1), 316–339.

Bayati, M., Kim, J. H., & Saberi, A. (2010). A sequential algorithm for generating random graphs. *Algorithmica*, *58*(4), 860–910.

Beichl, I., & Sullivan, F. (1999). Approximating the permanent via importance sampling with application to the dimer covering problem. *J. Comput. Phys.*, *149*(1), 128–147.

Békéssy, A., Bekessy, P., & Komlós, J. (1972). Asymptotic enumeration of regular matrices. *Stud. Sci. Math. Hungar.*, *7*, 343–353.

Besag, J., & Clifford, P. (1989). Generalized Monte Carlo significance tests. *Biometrika*, *76*(4), 633–642.

Bezáková, I., Bhatnagar, N., & Vigoda, E. (2007). Sampling binary contingency tables with a greedy start. *Random Struct. Algor.*, *30*, 168–205.

Bezáková, I., Sinclair, A., Štefankovič, D., & Vigoda, E. (2006). Negative examples for sequential importance sampling of binary contingency tables. In Y. Azar & T. Erlebach (Eds.), *Algorithms – ESA 2006* (Vol. 4168, p. 136-147). Berlin/Heidelberg: Springer.

Blanchet, J. H. (2009). Efficient importance sampling for binary contingency tables. *Ann. Appl. Probab.*, *19*(3), 949–982.

Brazzale, A. R. (2005). hoa: An R package bundle for higher order likelihood inference. *Rnews*, *5*, 20–27. (ISSN 609-3631)

Brazzale, A. R., & Davison, A. C. (2008). Accurate parametric inference for small samples. *Statist. Sci.*, *23*(4), 465–484.

Canfield, E. R., Greenhill, C., & McKay, B. D. (2008). Asymptotic enumeration of dense 0–1 matrices with specified line sums. *J. Comb. Theory A*, *115*(1), 32–66.

Chen, Y. (2006). Simple existence conditions for zero-one matrices with at most one structural zero in each row and column. *Discrete Math.*, *306*(22), 2870–2877.

Chen, Y. (2007). Conditional inference on tables with structural zeros. *J. Comput. Graph. Stat.*, *16*(2), 445–467.

Chen, Y., Diaconis, P., Holmes, S. P., & Liu, J. S. (2005). Sequential Monte Carlo methods for statistical analysis of tables. *J. Am. Statist. Assoc.*, *100*(469), 109–120.

Chen, Y., & Small, D. (2005). Exact tests for the Rasch model via sequential importance sampling. *Psychometrika*, *70*(1), 11–30.

Connor, E. F., & Simberloff, D. (1979). The assembly of species communities: chance or competition? *Ecology*, *60*, 1132–1140.

Cox, D. R. (1958). The regression analysis of binary sequences. *J. R. Statist. Soc. B*, *20*, 215–242.

Cytel. (2010). *LogXact 9*. Cambridge, MA: Cytel Inc.

Diaconis, P., & Evans, S. N. (2000). Immanants and finite point processes. *J. Comb. Theory A*, *91*(1-2), 305–321.

Durrett, R. (2010). *Probability: theory and examples* (4th ed.). New York: Cambridge Univ. Pr.

Fienberg, S., Meyer, M., & Wasserman, S. (1985). Statistical analysis of multiple sociometric relations. *Journal of the American Statistical Association*, *80*(389), 51–67.

Frey, B. J. (1998). *Graphical models for machine learning and digital communication*. Cambridge, MA: MIT Press.

Gale, D. (1957). A theorem on flows in networks. *Pac. J. Math.*, *7*, 1073–1082.

Goldenberg, A., Zheng, A., Fienberg, S., & Airoldi, E. (2010). A survey of statistical network models. *Foundations and Trends in Machine Learning*, *2*(2), 129–233.

Gotelli, N. J. (2000). Null model analysis of species co-occurrence patterns. *Ecology*, *81*(9), 2606–2621.

Greenhill, C., & McKay, B. D. (2009). Random dense bipartite graphs and directed graphs with specified degrees. *Random Struct. Algor.*, *35*(2), 222–249.

Greenhill, C., McKay, B. D., & Wang, X. (2006). Asymptotic enumeration of sparse 0–1 matrices with irregular row and column sums. *J. Comb. Theory A*, *113*(2), 291–324.

Harrison, M. T. (2012). Conservative hypothesis tests and confidence intervals using importance sampling. *Biometrika*, *99*(1), 57–69.

Harrison, M. T., & Geman, S. (2009). A rate and history-preserving resampling algorithm for neural spike trains. *Neural comput.*, *21*(5), 1244–1258.

Holland, P. W., & Leinhardt, S. (1981). An exponential family of probability distributions for directed graphs. *J. Am. Statist. Assoc.*, *76*, 33–50.

Jerrum, M., Sinclair, A., & Vigoda, E. (2004). A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *J. Assoc. Comp. Mach.*, *51*(4), 671–697.

Kannan, R., Tetali, P., & Vempala, S. (1999). Simple Markov-chain algorithms for generating bipartite graphs and tournaments. *Random Struct. Algor.*, *14*(4), 293–308.

Kong, A., Liu, J. S., & Wong, W. H. (1994). Sequential imputations and Bayesian missing data problems. *J. Am. Statist. Assoc.*, *89*, 278–288.

Kou, S. C., & McCullagh, P. (2009). Approximating the α-permanent. *Biometrika*, *96*(3), 635–644.

Littlewood, D. (1950). *The theory of group characters and matrix representations of groups*. Oxford Univ. Press.

Liu, J. S. (2001). *Monte Carlo strategies in scientific computing*. New York: Springer.

Macchi, O. (1975). The coincidence approach to stochastic point processes. *Adv. Appl. Probab.*, *7*, 83–122.

Matsumoto, M., & Nishimura, T. (1998). Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comp. Simul.*, *8*(1), 3–30.

McCullagh, P., & Møller, J. (2006). The permanental process. *Adv. Appl. Probab.*, *38*, 873–888.

McKay, B. D. (1984). Asymptotics for 0-1 matrices with prescribed line sums. In D. M. Jackson & S. A. Vanstone (Eds.), *Enumeration and design* (pp. 225–238). Academic Press.

McKay, B. D., & Wormald, N. C. (1990). Uniform generation of random regular graphs of moderate degree. *Journal of Algorithms*, *11*(1), 52–67.

Mehta, C. R., & Patel, N. R. (1995). Exact logistic regression: theory and examples. *Statist. Med.*, *14*(19), 2143–2160.

O'Neil, P. E. (1969). Asymptotics and random matrices with row-sum and column-sum restrictions. *B. Am. Math. Soc.*, *75*, 1276–1282.

Park, S. K., & Miller, K. W. (1988). Random number generators: good ones are hard to find. *Commun. Assoc. Comp. Mach.*, *31*(10), 1192–1201.

Ponocny, I. (2001). Nonparametric goodness-of-fit tests for the Rasch model. *Psychometrika*, *66*(3), 437–459.

Rao, A., Jana, R., & Bandyopadhyay, S. (1996). A Markov chain Monte Carlo method for generating random (0, 1)-matrices with given marginals. *Sankhyā: The Indian Journal of Statistics, Series A*, 225–242.

Rasch, G. (1960). *Probabilistic models for some intelligence and attainment tests.* Copenhagen: Danmarks Paedagogiske Institut.

Rasch, G. (1961). On general laws and the meaning of measurement in psychology. In J. Neyman (Ed.), *Proceedings of the fourth berkeley symposium on mathematical statistics and probability: Probability theory* (Vol. 4, pp. 321–334). Berkeley, CA.

Rothblum, U. G., & Schneider, H. (1989). Scalings of matrices which have prespecified row sums and column sums via optimization. *Linear Algebra Appl.*, *114*, 737–764.

Ryser, H. J. (1957). Combinatorial properties of matrices of zeros and ones. *Can. J. Math.*, *9*, 371–377.

Shirai, T., & Takahashi, Y. (2003). Random point fields associated with certain Fredholm determinants I: fermion, Poisson and boson point processes. *J. Funct. Anal.*, *205*(2), 414–463.

Sinkhorn, R. (1964). A relationship between arbitrary positive matrices and doubly stochastic matrices. *Ann. Math. Statist.*, *35*(2), 876–879.

Sinkhorn, R. (1967). Diagonal equivalence to matrices with prescribed row and column sums. *Am. Math. Mon.*, *74*(4), 402–405.

Snijders, T. A. B. (1991). Enumeration and simulation methods for 0–1 matrices with given marginals. *Psychometrika*, *56*(3), 397–417.

StataCorp. (2009). *Stata statistical software: Release 11.* College Station, TX: StataCorp LP.

Valiant, L. G. (1979). The complexity of computing the permanent. *Theor. Comput. Sci.*, *8*(2), 189–201.

Vaughan, R. J., & Venables, W. N. (1972). Permanent expressions for order statistic densities. *J. R. Statist. Soc. B*, *34*, 308–310.

Vere-Jones, D. (1988). A generalization of permanents and determinants. *Linear Algebra Appl.*, *111*, 119–124.

Vere-Jones, D. (1997). Alpha-permanents and their applications to multivariate gamma, negative binomial and ordinary binomial distributions. *New Zeal. J. Math.*, *26*, 125–149.

Verhelst, N. D. (2008). An efficient MCMC algorithm to sample binary matrices with fixed marginals. *Psychometrika*, *73*(4), 705–728.

Wasserman, S. S. (1977). Random directed graph distributions and the triad census in social networks. *J. Math. Sociol.*, *5*(1), 61–86.

Zamar, D., McNeney, B., & Graham, J. (2007). elrm: Software implementing exact-like inference for logistic regression models. *J. Statist. Software*, *21*, 1–18.